N° d'ordre : 4500 ANNÉE 2012

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : INFORMATIQUE

Ecole doctorale Matisse

présentée par

Nicolas Guillermin

préparée à l'UMR CNRS 6625
(Institut de Mathématique de Rennes)

Intitulé de la thèse :
Implémentation matérielle de coprocesseurs haute performance pour la cryptographie asymétrique

Thèse soutenue à Rennes le 6 janvier 2012

devant le jury composé de :

Jean-Claude BAJARD
Professeur au LIP6 / rapporteur

Jean-Luc BEUCHAT
Professeur associé, Université de Tsukuba / rapporteur

Ingrid VERBAUWHEDE
Professeur, KU Leuven / examinateur

Emmanuel PROUFF
Chercheur, Oberthur Technologies / examinateur

Reynald LERCIER
Chercheur, DGA.MI / examinateur

Sylvain DUQUESNE
Professeur à l'IRMAR / directeur de thèse

Résumé

Dans cette thèse, je propose des architectures de coprocesseurs haute performance pour implémenter les primitives de cryptographie asymétrique, comme le RSA, les courbes elliptiques ou le couplage. Les coprocesseurs décrits dans cette thèse ont été implémentés dans des FPGA, et présentent des performances jamais égalées auparavant dans la littérature publique sur ce type de technologie. La particularité de ces architectures est l'utilisation du Residue Number System, un mode de représentation alternatif qui utilise les restes chinois pour calculer efficacement les opérations arithmétiques sur les grands nombres. Ces travaux permettent de confirmer expérimentalement les avantages théoriques de ce mode de représentation pour l'arithmétique modulaire, issus de [14, 13, 43]. Au bénéfice théorique que le RNS apporte s'ajoute une forte capacité de parallélisation qui permet d'obtenir des designs réguliers et pipelinés, proposant une fréquence maximale importante tout en réalisant les opérations modulaires dans un nombre très faible de cycles, et ce quelle que soit la taille des nombres. A titre d'exemple, une multiplication scalaire sur une courbe de 160 bits s'effectue en 0.57 ms sur un Altera Stratix, et en 4 ms pour une courbe de 512 bits, là ou les techniques de représentation classiques réalisent la même opération en le double de temps, à technologie équivalente (excepté pour des courbes particulières). Dans le cas du couplage, le gain est encore plus intéressant, puisqu'il a permis une division par 4 de latence de la meilleure implémentation sur corps de grande caractéristique au moment de la publication de [35], et la première implémentation d'un couplage à 128 bits de sécurité sur corps de grande caractéristique à descendre en dessous de la milliseconde. Enfin, je démontre la capacité du RNS à sécuriser une implémentation haute performance, en proposant 2 contre-mesures contre les canaux auxiliaires et les fautes s'adaptant efficacement sur les coprocesseurs et pouvant être utilisées pour toutes les primitives cryptographiques basées sur l'arithmétique modulaire de grands nombres.

Abstract

In this PhD thesis I propose coprocessors architectures for high performance computations of asymmetric primitives like RSA, Elliptic Curves and Pairing. Coprocessors have been implemented in FPGA, and propose the lowest latency ever showed in public litterature on such targets. The novelty of these architectures is the usage of the Residue Number System (RNS), an alternate way to represent big numbers. The work presented here confirms with experimentation the theoretical advantages of this system previously emphasized by [14, 13, 43]. Together with this theoretical advantage RNS computation can be efficiently parallelized, and getting highly regular and parallelized architectures to reach high frequency while computing modular operations in few cycles is possible, whatever is the size of the numbers. For example, a scalar multiplication on a generic 160 elleptic curve can be executed in 0.57 ms on an Altera Stratix, and in 4 ms on a 512 bits curve, compared with classical representations which hardly do the same in twice this time with comparable technologies (except for particular curves). For Pairing the results are even more interesting, since a 4 times division of the latency had been reached by the time [35] was published, and the first time a Pairing over large characteristic fields was executed in less than 1 ms on a FPGA. Eventually, I demonstrate the ability RNS to provide original solutions to protect computations against side channel and perturbation threats. I propose 2 countermeasures to thwart faults and power analysis which can be used on every primitives relying on big number modular arithmetic. These countermeasures are designed to be efficiently adapted on the RNS coprocessors.

Table des matières

1 Conception VLSI, arithmétique modulaire et sécurité 11

1.1 Introduction générale à la conception VLSI . . . 12
1.1.1 Du transistor au schéma de circuit . . . 12
1.1.2 Du schéma de circuit aux langages HDL . . . 13
1.1.3 Développement sur plate forme FPGA . . . 14
1.1.4 Efficacité et chemin critique . . . 16
1.2 Etat de l'art des implémentations efficaces de l'arithmétique modulaire sur des grands nombres . . . 18
1.2.1 Nombres de Mersenne et Pseudo-Mersenne . . . 18
1.2.2 Réduction modulaire de Barrett . . . 19
1.2.3 Montgomery . . . 19
1.2.4 Algorithmes de multiplication scalaire ou d'exponentiation . . . 20
1.2.5 Le RSA et l'implémentation RSA-CRT . . . 22
1.3 Attaques physiques sur les composants . . . 22
1.3.1 Analyse de canaux auxiliaires . . . 23
1.3.2 Attaques par fautes . . . 25

2 Le Residue Number System : théorie et implémentation efficace 27

2.1 Introduction . . . 27
2.2 Le Residue Number System . . . 28
2.2.1 Le théorème des restes chinois . . . 28
2.2.2 Application à un système de notation des nombres : le RNS . . . 29
2.3 RNS et arithmétique modulaire . . . 30
2.3.1 Adaptation de Montgomery pour le RNS . . . 30
2.3.2 Utilisation du Mixed Radix System . . . 31
2.3.3 Transformations de Shenoy et Kumaresan . . . 32
2.3.4 Approximation dans le changement de base : transformations de Posch et Kawamura . . . 33
2.3.5 Méthode agressive : l'amélioration de Bajard et al. . . . 34
2.4 Architecture Cox-Rower . . . 35
2.4.1 Architecture générale du Cox-Rower . . . 35
2.4.2 Cox . . . 37
2.4.3 Rower . . . 38
2.4.4 Séquencement des changements de base et besoin en mémoires et précalculs . . . 39
2.4.5 Pilotage du Cox-Rower . . . 40
2.5 Transformations binaire → RNS et RNS → binaire . . . 41
2.5.1 Transformation binaire → RNS . . . 42
2.5.2 Transformation RNS → binaire . . . 42

3 Implémentation efficace de la multiplication scalaire sur courbe elliptique en RNS à tous les niveaux de sécurité 45
3.1 Introduction . . . 45
3.2 Arithmétique des courbes elliptiques, impacts du RNS . . . 46
3.2.1 Courbes elliptiques : définition . . . 46
3.2.2 Diviseurs et structure de groupe . . . 47
3.2.3 Formules d'addition et doublement sur une courbe elliptique définie sur $\mathbb{F}_p$ . . . 48
3.2.4 Impact du RNS sur l'arithmétique des courbes . . . 50
3.2.5 Choix de base . . . 51
3.3 Parallélisation et pipeline . . . 52
3.3.1 Optimisation des variables temporaires, dépendance de données et parallélisation . . . 52
3.3.2 Structure du pipeline . . . 54
3.3.3 Sécurité . . . 57
3.3.4 Conclusion . . . 58
3.4 Performances et comparaisons . . . 58
3.4.1 Résultat d'implémentation pour l'ensemble des niveaux de sécurité . 58
3.4.2 Comparaisons . . . 59
3.5 Conclusion . . . 61

4 Implémentation efficace de couplage sur les courbes Barreto-Naehrig 62
4.1 Introduction . . . 62
4.2 Couplage sur les courbes elliptiques . . . 63
4.2.1 Définition et motivation . . . 63
4.2.2 Fonctions de Miller et couplage de Tate . . . 65
4.2.3 Couplages de Ate et Optimal Ate . . . 66
4.2.4 Les courbes Barreto-Naehrig . . . 66
4.2.5 Choix de courbes et extension de corps . . . 68
4.3 Calcul de couplage avec un Cox-Rower . . . 68
4.3.1 Paramétrisation du Cox-Rower pour le couplage . . . 69
4.3.2 Architecture du pipeline . . . 69
4.4 Arithmétique sur les extensions de $\mathbb{F}_p$ . . . 71
4.4.1 Arithmétique sur $\mathbb{F}_{p^2}$: retour à la méthode naïve . . . 71
4.4.2 Arithmétique sur $\mathbb{F}_{p^{12}}$, étude de l'apport des techniques d'interpolation 71
4.4.3 Inversion sur $\mathbb{F}_{p^{12}}$ . . . 73
4.4.4 Séquencement de plus haut niveau . . . 73
4.4.5 Contrôle des variables locales . . . 75
4.5 Résultats d'implémentation, analyse et comparaison . . . 76
4.5.1 Taille du circuit . . . 76
4.5.2 Latence d'un couplage . . . 76
4.5.3 Comparaisons et discussions . . . 77
4.6 Conclusion . . . 77
4.7 Annexe A : circuit optimisé pour la courbe BN_{126} sur FPGA Virtex 6 . . . 79

5 Le RNS : une solution efficace contre les attaques par canaux auxiliaires et par perturbations 82
5.1 Introduction . . . 82
5.2 Implémentation efficace de la contre-mesure de Bajard et al. [16] . . . 83

5.2.1 La contre-mesure LRA 84
5.2.2 Transfert de mémoire entre Rowers 85
5.2.3 Précalculs 86
5.2.4 Impacts sur l'architecture du Cox-Rower 88
5.3 La contre-mesure de détection de faute pas le RNS (contre-mesure RFD) . . 88
5.3.1 Modèle de faute 88
5.3.2 Le s-Cox : utilisation d'un développement limité à l'ordre s 89
5.3.3 Détection de faute via l'utilisation du s-Cox 89
5.3.4 Détection d'erreur sur la base $\mathcal{B}'$ 92
5.3.5 Perturbation sur la base $\mathcal{B}$ 93
5.3.6 Perturbation sur le s-Cox ou sur le séquenceur 94
5.3.7 Combinaison des 2 contre-mesures 94
5.4 Implémentation d'un RSA-CRT 1024 protégé avec les contre-mesures LRA et RFD 95
5.4.1 Dimensionnement du Cox-Rower pour l'exécution du RSA-CRT . . . 95
5.4.2 Calcul du représentant de Montgomery 97
5.4.3 Reconstruction du CRT et transformation RNS $\rightarrow$ binaire 98
5.4.4 Latence du RSA-CRT 98
5.5 Conclusion 98
5.6 Annexe A : La contre-mesure de Ciet et al. 100

Table des figures

1.1 Schéma des portes logiques utilisées dans cette thèse 12
1.2 Une bascule : le signal r prend la valeur d au front montant du signal d'horloge h 13
1.3 Exemple : définition d'un registre en VHDL 13
1.4 Un élément logique de FPGA : le contenu de la RAM et la valeur statique du multiplexeur configurent une porte logique 15
1.5 Une ALM de Stratix III : 16
1.6 Automate de Moore 17
1.7 Une attaque SPA réussie, la clé est lue directement sur le canal de fuite . . 24

2.1 Architecture générale du Cox-Rower 36
2.2 Le Cox : un accumulateur à démarrage piloté 38
2.3 Le Rower : unité de traitement et de stockage des variables temporaires . . 40
2.4 Module de pilotage du Cox-Rower : un séquenceur spécifiquement dessiné pour les opérations cryptographiques 44

3.1 Addition sur une courbe elliptique définie sur les réels 49
3.2 Dépendance des variables temporaires et parallélisme 53
3.3 Structure de Rower pour les courbes elliptiques 55

4.1 Rower pour le couplage et pipeline 81

5.1 Réseau de Benz $n = 8$ 87
5.2 Architecture de Cox à 2 étages 90
5.3 Architecture de Cox à $l > 2$ étages 91
5.4 Architecture du Rower et pipeline à 7 étages 96

Liste des tableaux

1.1 Noms des gammes de produit Xilinx et Altera, et nœud technologique correspondant 15
1.2 Types de mémoires de la famille Stratix et configurations possibles 17

2.1 Liste des précalculs nécessaires au changement de base, aux transformations binaire $\leftrightarrow$ RNS 42

3.1 Formules d'addition et doublement sans l'ordonnée du point 49
3.2 Formules des lois de groupe limitant le nombre de réductions 51
3.3 Dimensionnement des bases RNS en fonction du niveau de sécurité souhaité 52
3.4 Structure du pipeline 56
3.5 Résultats détaillés sur Altera Stratix et Stratix II 59
3.6 Comparaison de notre implémentation avec d'autres contributions 61

4.1 Description du pipeline et comparaison avec le chapitre 3 70
4.2 Consommation des ressources des FPGA utilisés 76
4.3 Coût des opérations intermédiaires dans le couplage optimal Ate 77
4.4 État de l'art des implémentations orientées vitesse des couplages 78

5.1 Liste des précalculs à transmettre pour un canal m_i de $\mathcal{M}(\mathcal{B})$ 85
5.2 Consommation des ressources du FPGA 101
5.3 Temps de calcul pour un RSA-CRT avec et sans les contre-mesures 101

Liste des Algorithmes

1 Réduction modulaire par un nombre de pseudo-Mersenne 19
2 Algorithme de Montgomery 19
3 Simple Operand Scanning 20
4 Coarsely Integrated Operand Scanning 20
5 Most Significant Bit 21
6 Least Significant Bit 21
7 W-Slinding Windows 21
8 Non adjacent Form 21
9 Implémentation RSA-CRT 22
10 Echelle de Montgomery et contre-mesure de Giraud [57] 24
11 $\texttt{RedM}(X, p, \mathcal{B}, \mathcal{B}')$ 30
12 RNS2$MRS(X, \mathcal{B}_1)$ 32
13 Algorithme de changement de base de Kawamura 35
14 Algorithme de réduction de Montgomery en $\eta(2n + 3)$ cycles sur un Cox-Rower de rapport η. Chaque opération est exécutée sur chacun des Rower sur η cycles pour le traitement sur la base complète. 41
15 Algorithme de transformation binaire vers RNS 42
16 Algorithme de transformation RNS vers binaire 43
17 $Montg - ladder(k, \mathbf{G}, \mathbf{C}_{p,a_4,a_6})$ 50
18 Formules pour obtenir les coordonnées affines de $k\mathbf{G}$ à l'issue de l'algorithme d'exponentation 51
19 Additionneur modulaire 56
20 Algorithme de Miller 65
21 Calcul de carré dans le sous groupe cyclotomique de $\mathbb{F}_{p^{12}}$ 72
22 Carré durant l'exponentiation finale [60] 72
23 Calcul de f^{p^6-1} 73
24 Couplage Optimal Ate pour les courbes BN pour $x < 0$ 74
25 `dbl`, doublement 74
26 `add`, addition 74
27 `hard-part`, partie difficile de l'exponentiation finale proposée par [112] 75
28 $RSA - CRT$ for RNS (p, q, d_p, d_q, X, m) 95
29 $\texttt{TrMgt}(m, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$ 97

Notation Par souci de clarté, dans l'ensemble de cette thèse $|a|_b = a \mod b$.

Introduction

La cryptographie, étymologiquement *écriture cachée*, est apparue dans l'antiquité et s'est progressivement développée parallèlement à la progression des communications militaires. En effet jusqu'au milieu des années 1970, son objectif se cantonnait à créer les outils permettant de transmettre sur des canaux publics de communication (radio par exemple) des informations confidentielles dans un objectif quasi-exclusivement militaire ou diplomatique.

L'augmentation des performances de calcul et des débits de télécommunication a permis l'émergence de la cryptographie moderne. Le développement de domaines comme le commerce électronique et le contrôle d'accès a progressivement élargi la cryptographie au-delà de son objectif historique qui est la protection d'informations sensibles. La cryptographie moderne propose ainsi des solutions nécessaires à l'authentification de personnes ou de données, au contrôle d'accès, à la signatures voire au vote électronique.

La science de la cryptographie a en partie pour but d'apporter les garanties les plus fortes possibles sur la sécurité des schémas. Pour cela, les cryptologues utilisent une modélisation de la réalité, où les protocoles cryptographiques sont exécutés par des machines de Turing mettant en œuvre des informations, dont certaines sont publiques (les données chiffrées par exemple), et d'autres privées dont la divulgation met en péril le service apporté. Ce sont par exemple, les données non chiffrées, mais aussi les clés cryptographiques, ou des données aléatoires. De manière plus imagée, on peut considérer que les acteurs des protocoles sont enfermés avec leur secret dans des coffres-forts, et que l'attaquant ne peut accéder qu'aux interfaces de ceux-ci. Dans ce type de modèle, la sécurité des services apportés par la cryptographie peut être démontrée, à partir de propriétés admises sur les opérations de base que l'on appelle primitives cryptographiques, voire sur des sous parties de ces primitives.

Quand on replace la cryptographie dans la réalité, une problématique importante est donc la définition d'un coffre-fort, qui doit en même temps être robuste aux tentatives d'intrusion, avoir les capacités suffisantes pour calculer les données publiques des protocoles, et suffisamment peu onéreux pour pouvoir en mettre dans toutes les poches. C'est ainsi qu'on a vu l'émergence de la carte à puce, ou carte à microprocesseur, dont la spécificité est la capacité (tout juste suffisante) de réaliser des calculs cryptographiques, tout en préservant en son sein les secrets nécessaires à la réalisation des protocoles cryptographiques.

Cette thèse étudie des pistes nouvelles pour l'implémentation efficace et sûre des primitives cryptographiques asymétriques les plus utilisées à l'heure actuelle (RSA, courbes elliptiques). En effet, l'intégration de ce type de primitives dans des produits à forte contrainte de sécurité (et donc utilisant des composants cryptographiques) reste encore aujourd'hui un défi pour l'industrie, et un sujet de recherche très dynamique. En outre, ces primitives mettent toutes les deux en œuvre une brique de base qui est l'arithmétique modulaire sur les grands nombres.

L'objectif poursuivi dans cette thèse est la recherche d'implémentations sécurisées haute

performance. Les circuits proposés dans cette thèse visent ainsi des domaines où la latence des calculs de primitives asymétriques doit être aussi faible que possible, tout en étant très sécurisés vis-à-vis des attaques (même locales). J'étudie l'apport d'une technique de représentation des nombres alternative, appelée Residue Number System. Par une recherche d'optimisations et une démarche expérimentale, je démontre que cette technique offre des résultats excellents pour un développeur recherchant sécurité, flexibilité et efficacité.

La structure de cette thèse est la suivante : le chapitre 1 introduit l'ensemble des concepts nécessaires à une bonne compréhension de tous les chapitres de cette thèse. Sont abordés la conception VLSI, les techniques classiques d'implémentation de l'arithmétique modulaire des grands nombres, ainsi que les concepts traditionnels de la sécurité matérielle. Le chapitre 2 introduit ce qu'est le Residue Number System, son utilisation pour l'arithmétique modulaire, et décrit les éléments communs des coprocesseurs RNS qui seront utilisés dans les chapitre 3,4 et 5. Le chapitre 3 décrit un coprocesseur scalable pour la réalisation de la multiplication scalaire sur courbes elliptiques définies sur des corps premiers, permettant de traiter n'importe quelle courbe elliptique à un niveau de sécurité donné. Une brève introduction aux courbes elliptiques y est donnée. Le chapitre 4 adapte les travaux du chapitre 3 pour proposer l'implémentation la plus rapide jamais proposée pour un couplage sur FPGA à 128 bits de sécurité sur une courbe définie sur un corps de grande caractéristique. Je propose aussi la première implémentation matérielle d'un couplage présentant 192 bits de sécurité. Ce chapitre commence par une introduction sur les couplages sur courbes elliptiques.

Enfin le chapitre 5 propose une amélioration de la contre-mesure proposée par [16] pour l'exécution dans un contexte de haute performance, ainsi qu'une nouvelle contre-mesure contre les perturbations garantissant une sécurisation d'une grande partie du circuit contre une large variété de perturbations, et ce pour toutes les primitives utilisant l'arithmétique modulaire sur des grands nombres, comme le RSA, le DSA et les courbes elliptiques.

Chapitre 1

Conception VLSI, arithmétique modulaire et sécurité

Sommaire

1.1 Introduction générale à la conception VLSI 12
1.1.1 Du transistor au schéma de circuit 12
1.1.2 Du schéma de circuit aux langages HDL 13
1.1.3 Développement sur plate forme FPGA 14
1.1.4 Efficacité et chemin critique 16
1.2 Etat de l'art des implémentations efficaces de l'arithmétique modulaire sur des grands nombres 18
1.2.1 Nombres de Mersenne et Pseudo-Mersenne 18
1.2.2 Réduction modulaire de Barrett 19
1.2.3 Montgomery 19
1.2.4 Algorithmes de multiplication scalaire ou d'exponentiation 20
1.2.5 Le RSA et l'implémentation RSA-CRT 22
1.3 Attaques physiques sur les composants 22
1.3.1 Analyse de canaux auxiliaires 23
1.3.2 Attaques par fautes 25

Introduction

Dans ce chapitre j'introduis les différents concepts utilisés dans cette thèse. La section 1.1 décrit succinctement la conception VLSI, qui a été largement utilisée durant cette thèse. En effet, l'ensemble des circuits que je propose est en grande partie implémenté en VHDL, avec une utilisation parcimonieuse des générateurs spécifiques à chaque FPGA, afin de garantir la portabilité de mes codes sur une large variété de FPGA. Dans la section 1.2, je rappelle les différentes techniques disponibles dans la littérature pour implémenter la réduction modulaire sur des grands nombres, opération qui est une brique de base de toutes les primitives cryptographiques utilisées dans cette thèse. Enfin, j'introduis les techniques à l'état de l'art permettant d'exploiter les canaux auxiliaires et les perturbations dans la section 1.3. Pour démontrer la puissance de ce type d'attaque, je les illustrerai sur une implémentation RSA-CRT, que j'introduirai préalablement dans la section 1.2.5.

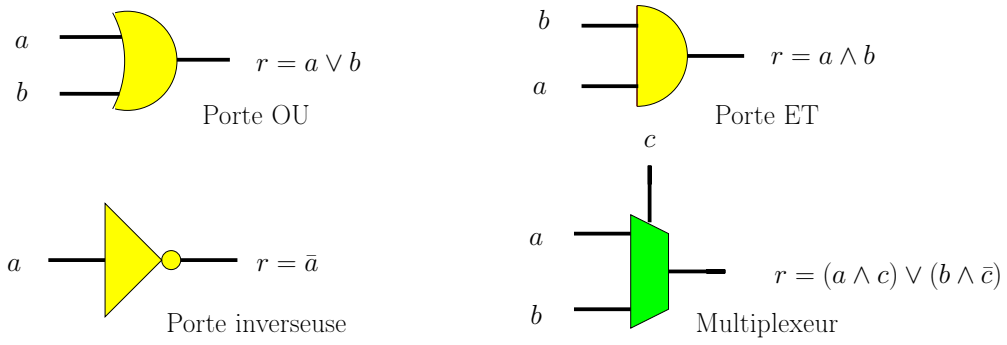
1.1 Introduction générale à la conception VLSI

1.1.1 Du transistor au schéma de circuit

La conception VLSI (pour Very Large Scale Integration) est apparue conjointement avec la miniaturisation de la microélectronique. En effet, le nombre de transistors augmentant continuellement en suivant la loi de Moore, il est apparu nécessaire que le développement de ces composants bénéficie de gains de productivité. Or, si le nombre de transistors double tous les 18 mois, il n'est pas question que le temps de développement double tous les 18 mois lui aussi. Il a donc été nécessaire de créer des méthodes permettant au développeur de s'abstraire du circuit réel, pour se consacrer uniquement à la fonctionnalité à réaliser, au même titre que l'avènement des langages de programmation et leur différents paradigmes a permis de gagner en productivité dans le domaine du génie logiciel.

Une première abstraction, appelée schéma de circuit, permet de dessiner un circuit à partir de fonctionnalités prédéfinies, soit dans le domaine du circuit digital un assemblage de transistors P et N prédéfinis, et permettant de réaliser une fonction booléenne. Cet assemblage est appelé porte, et la figure 1.1 liste celles qui apparaîtront dans les schémas de cette thèse.

FIGURE 1.1 – Schéma des portes logiques utilisées dans cette thèse

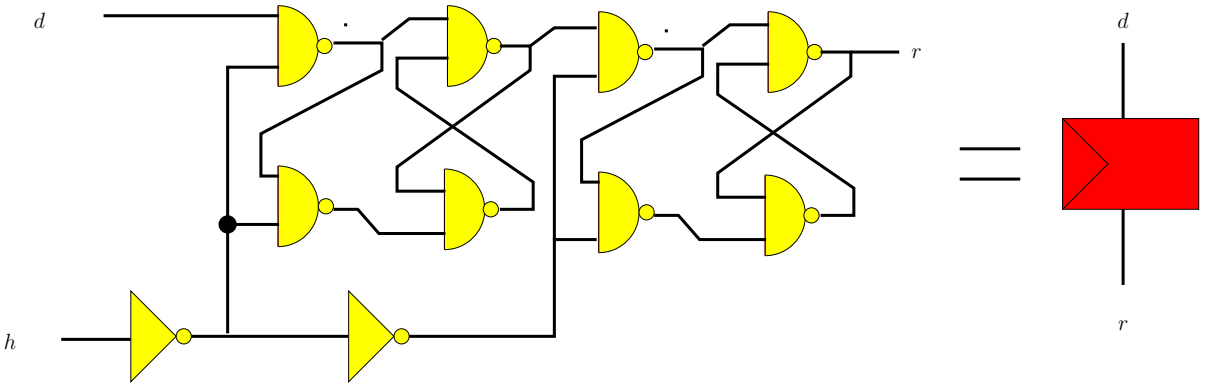


Par la suite, ces portes peuvent être assemblées pour obtenir des circuits plus complexes. L'exemple de la figure 1.2 donne le schéma d'une bascule D avec signal de réinitialisation (dit signal de reset), comme un assemblage de de portes nand et inverseuses et dans un format simplifié, telle qu'on la retrouvera tout le long de cette thèse (le signal d'horloge h y est sous-entendu).

Dans le reste de cette thèse je ne travaille que sur des circuits cadencés par une unique horloge, dits circuits synchrones. Ainsi, les éléments constituant un circuit appartiennent à une des 2 catégories suivantes :

- des fonctions combinatoires sans mémoire, qui ne sont pas cadencées par l'horloge. La valeur de sortie de ces portes suit quasi instantanément la valeur de ses entrées.
- des registres, qui sont des éléments de mémorisation cadencés par une horloge, et qui ne disposent que de 2 entrées, la valeur à mémoriser et l'horloge (une troisième peut être ajoutée qui permet une réinitialisation de la valeur mémorisée de manière asynchrone).

FIGURE 1.2 – Une bascule : le signal r prend la valeur d au front montant du signal d'horloge h



1.1.2 Du schéma de circuit aux langages HDL

Dans les années 1980 est apparue la nécessité d'élever encore le niveau d'abstraction afin d'accélérer le développement des composants. Ce besoin s'est caractérisé par 2 initiatives parallèles, donnant naissance à 2 langages concurrents encore massivement utilisés aujourd'hui :

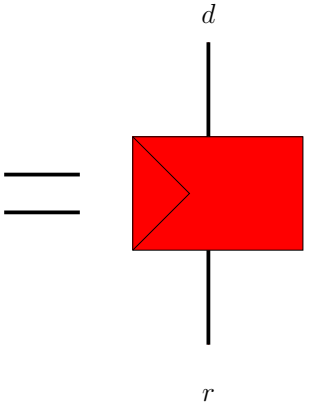
- une initiative privée de la société Cadence Design Systems [2] qui développa un langage propriétaire appelé Verilog, et aujourd'hui standardisée par la norme IEEE 1364 [5].
- une initiative lancée par le Département de la Défense des États-Unis, qui a donné naissance au VHSIC Hardware Descritpion Langage (VHDL), aujourd'hui normalisé (norme IEEE 1076[6]).

La figure 1.3 donne une description de la même bascule que la figure 1.2 en VHDL. On peut constater que pour le développeur, la description est beaucoup plus simple.

FIGURE 1.3 – Exemple : définition d'un registre en VHDL

```
entity bascule is
  port(
    d : in std_logic;
    h : in std_logic;
    r : out std_logic
  );
end bascule;

architecture arch of bascule is
begin
  r <= d when(h='1' and h'event);
end arch;
```



Le VHDL est un langage de programmation dont la syntaxe est proche de l'ADA, et dont la sémantique répond à un paradigme propice à la description d'un composant, com-

posé de fils portant les variables manipulées, et de portes, qui réalisent toutes en parallèle des opérations sur les valeurs portées par ces fils. Ainsi, le VHDL dispose de signaux correspondant aux fils, des processus correspondant aux portes logiques, et un séquenceur général gérant une variable de temps assure l'exécution parallèle des processus.

L'avantage d'un composant écrit en VHDL est qu'il peut être simulé, mais sous certaines conditions, il peut être aussi synthétisé, c'est-à-dire transformé en un schéma de circuit décrit ci-dessus. La condition est que le programme soit inclus dans un sous ensemble du VHDL appelé VHDL synthétisable, et décrit dans les norme IEEE 1076.3 et IEEE 1164 [6, 4]. Ainsi, un concepteur peut utiliser le même langage pour tester un composant avant que celui-ci n'existe, et obtenir lors d'une exécution la valeur de ses signaux internes (à la manière d'un debugger), et le synthétiser dans une description schématique, qui sera elle aussi transformée automatiquement, via un processus appelé placement-routage en une description du composant respectant la géométrie des éléments. Le VHDL est donc l'entrée du processus de fabrication pour un grand nombre de composants, notamment les composants numériques.

Je renvoie le lecteur aux références [69, 70], qui définit de manière claire et formelle une sémantique de simulation correspondant à une sous partie du VHDL, ou [74], qui travaille sur la sémantique de la synthèse.

1.1.3 Développement sur plate forme FPGA

Les "Field-Programmable Gate Array" ou FPGA sont une famille de composants programmables, dont le programme, appelé "bitstream" n'a pas vocation à être exécuté par un microprocesseur, mais à configurer des portes logiques et une logique d'interconnexion permettant de relier ces portes entre elles.

Les constructeurs principaux de FPGA sont deux sociétés américaines, Xilinx et Altera, disposant toutes deux de 2 types de composants : une catégorie appelée "low-cost" pour les applications de série (le coût de la puce reste suffisamment faible pour être intégré dans des équipements de grande série), une catégorie appelée "high end", dont les capacités sont plus importantes, et utiles pour les activités de prototypage ou pour les équipements de faible série (le coût unitaire pouvant atteindre quelques milliers de dollars). La différence entre les deux types de produits réside essentiellement dans leur taille, et parfois dans la nature des fonctions qu'ils embarquent. A chaque nœud technologique de fonderie correspond un produit dans chaque catégorie dont le tableau 1.1 rappelle les noms. On peut noter l'apparition d'une catégorie de FPGA de taille moyenne dite "midrange", dont le nom commercial est Kinect chez Xilinx et Arria chez Altera.

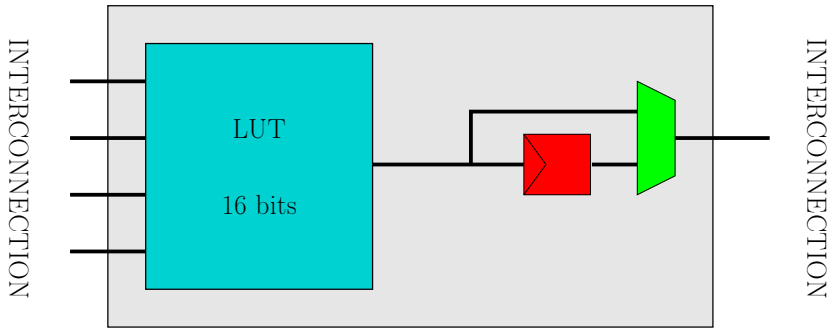
Les portes logiques sont constituées de mémoires rapides et de faible capacité appelées "Look-Up-Table", ou LUT. Historiquement chez la plupart des fabricants de FPGA une LUT contenait 16 bits, et permettait donc de réaliser n'importe quelle fonction combinatoire 4 vers 1. A chaque LUT peut être adjoint un registre piloté par une horloge et un multiplexeur permettant d'utiliser le registre ou non. La figure 1.4 représente une LUT simple. Chaque fabricant de FPGA a par la suite réalisé des choix de regroupement de LUTs, et d'ajout de capacités supplémentaires. A titre d'exemple, et parce que ce type de FPGA sera utilisé durant cette thèse, la figure 1.5 représente un "Altera Logic Module" ou ALM constituant la logique élémentaire du Stratix III. Ainsi, l'ALM est constitué d'une série de LUT pouvant être configuré de nombreuses façons. Le circuit marron permet une propagation de retenue rapide pour permettre d'accélérer l'arithmétique des nombres, tandis que le circuit bleu permet de générer aisément des registres à décalage. Pour plus de détails le lecteur peut se reporter au manuel [7]. L'équivalent chez Xilinx est le "slice",

TABLE 1.1 – Noms des gammes de produit Xilinx et Altera, et nœud technologique correspondant

Taille de grille	Xilinx		Altera	
	low cost	high end	low cost	high end
220 nm		Virtex		
180 nm		Virtex-E		
150 nm		Virtex-II		
130 nm	Spartan	Virtex-II pro	Cyclone	Stratix
90 nm	Spartan-2	Virtex-4	Cyclone II	Stratix II
65 nm	Spartan-3	Virtex-5	Cyclone III	Stratix III
40 nm	Spartan-6	Virtex-6	Cyclone IV	Stratix IV
28 nm	Artix	Virtex-7	Cyclone V	Stratix V

dont la capacité dépend elle aussi du nœud et de la gamme.

FIGURE 1.4 – Un élément logique de FPGA : le contenu de la RAM et la valeur statique du multiplexeur configurent une porte logique

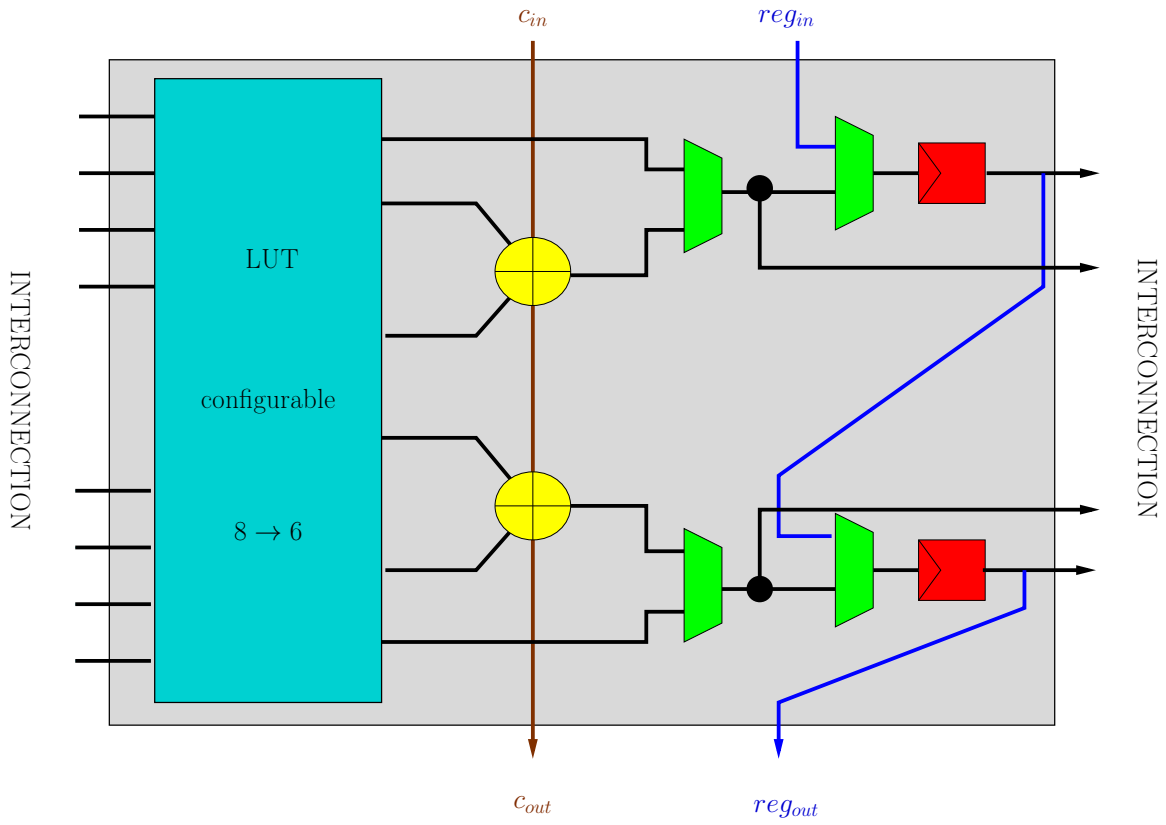


L'interconnexion est constituée d'une matrice dont la topologie dépend du fabricant, et dont l'objectif est le routage des signaux entre les différentes logiques élémentaires, mais aussi entre les différents blocs spécifiques se trouvant dans les FPGA modernes.

Si l'on prend l'exemple du Stratix III, 8 ALM sont regroupés dans un Logic Array Block (LAB) via une interconnexion locale haute performance. Ces LABs sont regroupés en une matrice d'interconnexion rapide 4×4 (incluant 128 ALM). Chacun de ces groupes est maillé dans une matrice d'interconnexion plus lente 5×3 (soit 1920 ALM). Ces blocs sont ensuite maillés dans une interconnexion globale lente et de taille variable. Cette taille dépend du modèle dans chaque technologie, et se trouve dans la nomenclature du composant. Par exemple, la série EP3S50 est la série de Stratix III dispose de 50000 ALM.

Toujours dans le Stratix III, certains des LABs sont avantageusement remplacés par des mémoires double port de différentes tailles, et dont la largeur du bus d'adresse est configurable. Le tableau 1.2 ci-dessous dresse la liste des mémoires et leurs différentes configurations. Comme on peut le constater sur les familles de FPGA Altera, ces mémoires sont construites sous un format de 9 bits au lieu de 8 pour les mémoires classiques, la

FIGURE 1.5 – Une ALM de Stratix III :



raison est la capacité d'utiliser ou non ce bit supplémentaire comme bit de parité. Comme le lecteur le verra par la suite, ce bit supplémentaire sera très utile pour les opérations cryptographiques étudiées dans cette thèse.

Il faut noter que le concurrent direct d'Altera, Xilinx propose aussi des mémoires taillées autour d'un mot de 9 bits.

Enfin, toutes les générations "high end" depuis le Stratix et le Virtex-II et certaines "low cost" disposent de multiplieurs intégrés. En effet, si l'utilisation des LUT est possible pour réaliser des multiplieurs, ceux-ci sont trop lents et trop consommateurs d'éléments de logique. Au vu des besoins des utilisateurs de FPGA, Xilinx et Altera ont intégré des multiplieurs câblés. Par la suite ces multiplieurs ont augmenté leur capacité pour devenir des blocs dits "Digital Signal Processing" (DSP) performants, capables de réaliser à très grande vitesse des multiplications accumulations, opérations nécessaires dans de nombreuses applications de traitement du signal, comme par exemple la FFT . La taille de ces multiplieurs peut être configurée par multiples de 9 bits en général (à l'exception du Virtex 6, qui propose une taille de 18×25 signé). Je réfère le lecteur aux manuels des fournisseurs des FPGA utilisés pour une description complète des capacités des blocs DSP [8, 1].

1.1.4 Efficacité et chemin critique

Dans cette thèse je ne considère que des circuits complètement synchrones, c'est à dire des circuits dont tous les registres sont cadencés par une unique horloge. Ces composants

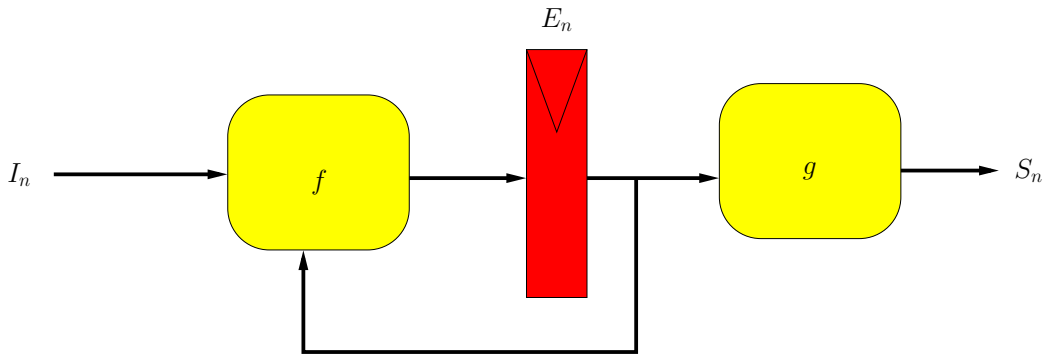
TABLE 1.2 – Types de mémoires de la famille Stratix et configurations possibles

Cyclone II	Stratix et Stratix II			Stratix III		
M4K	M512	M4K	M-RAM	MLAB	M9K	M144K
$4K \times 1$	512×1	$4K \times 1$	$64K \times 9$	64×8	$8K \times 1$	$16K \times 8$
$2K \times 2$	256×2	$2K \times 2$	$32K \times 18$	64×10	$8K \times 1$	$16K \times 9$
$1K \times 4$	128×4	$1K \times 4$	$16K \times 36$	32×20	$4K \times 2$	$8K \times 18$
512×9	64×9	512×9	$8K \times 72$		$2K \times 4$	$4K \times 36$
256×18	32×18	256×18	$4K \times 144$		$1K \times 9$	$2K \times 72$
128×36		128×36			512×18	
					256×36	

peuvent être ainsi considérés comme des automates de Mealy ou Moore [92, 97] en fonction du chemin liant les entrées aux sorties. Afin d'obtenir une valeur de chemin critique correspondant à l'ensemble du circuit, je ne m'intéresserai qu'aux automates de Moore.

Un automate de Moore est un quintuplet (I_n, S_n, O_n, f, g), tel que $I_n \in \mathbb{I}^{\mathbb{N}}$,$S_n \in \mathbb{S}^{\mathbb{N}}$,$O_n \in \mathbb{O}^{\mathbb{N}}$ où $\mathbb{I}$,$\mathbb{S}$,$\mathbb{O}$ sont des espaces finis appelés espace des entrées, des états et des sorties, et f et g deux fonctions $\mathbb{I} \times \mathbb{S} \to \mathbb{S}$ et $\mathbb{S} \to \mathbb{O}$ appelées fonction de transition et fonction de génération des sorties. Ainsi, tout circuit électronique synchrone correspond à la figure 1.6, ou f et g sont des fonctions combinatoires.

FIGURE 1.6 – Automate de Moore



Sur le composant physique, le temps de traversée de la fonction f par le signal électrique n'est pas immédiat. On peut considérer que E_n est constitué de k bits $E_{n,j}$, définissant k fonctions de transition f_j. Chaque f_j utilise un sous-ensemble de bits de C_j de $E_n \times I_n$ qui influent sur le résultat de f_j. Le doublet (C_j, f_j) est appelé cône logique de $E_{n,j}$. Après le front montant de l'horloge, le signal électrique doit tout d'abord être stabilisé sur chacun des éléments b de C_j. Cette phase est appelée phase de "setup", dont le temps est appelé "temps de setup" et est noté $T_s(b)$. Puis le temps de traversée de la fonction f par le signal électrique est appelé temps de traversée noté $T_t(C_j, b)$, car il est spécifique à chaque élément b de C_j. Enfin, une fois la sortie f stabilisée, un temps est nécessaire avant l'apparition du nouveau front montant de l'horloge pour que la mémorisation de $E_{n+1,j}$ se réalise correctement. Ce temps est appelé temps de "hold" et est noté T_j.

Les éléments j et b qui définissent la valeur maximale $max_{f_j}(max_{b \in C_j}(T_s(b)+T_t(C_j, b)+$

T_h) définissent le chemin critique du composant. Si b n'est pas une entrée (ce qui sera toujours le cas dans les circuits présentés dans cette thèse), alors cette valeur maximale définit la durée minimale du cycle d'horloge, et donc la fréquence maximale à laquelle le circuit peut fonctionner sans risque d'une erreur. Si cette fréquence maximale n'est pas respectée, alors il existe un risque que le composant physique ne respecte pas sa sémantique, et renvoie un résultat faux.

Ainsi, je donnerai pour chaque implémentation réelle sur un FPGA la fréquence maximale atteignable par le circuit, donnée par le logiciel de génération de bitstream que j'utiliserai dans cette thèse.

1.2 Etat de l'art des implémentations efficaces de l'arithmétique modulaire sur des grands nombres

Dans cette section, j'aborde les techniques classiques pour réaliser l'arithmétique modulo un entier p, quand p est un nombre impair de grande taille, et que le circuit permettant de traiter les opérations dispose d'un multiplieur permettant de traiter des mots de r bits. Cette valeur r est généralement égale à 8, 32 ou 64 bits dans les microprocesseurs et microcontrôleurs classiques.

L'algorithme naïf est en effet peu adapté à la structure des machines classiques, car celui-ci nécessite la réalisation de divisions euclidiennes, implémentées sur des grands nombres par l'algorithme "schoolbook" ou l'algorithme de Newton-Raphson [88], qui réalisent des estimations sur la valeur du quotient. Ces estimations, appelées "trial division" sont coûteuses en temps, et par ailleurs ne prennent pas en compte la nature statique de p. En effet dans les algorithmes cryptographiques, cette valeur reste constante pour chaque utilisateur (dans le cas du RSA), voire même commune à tous les utilisateurs (comme pour le DSA ou les courbes elliptiques). Cette propriété permet avantageusement de remplacer des calculs modulaires coûteux par des précalculs, certes d'un prix comparable voire supérieur, mais étant réalisés une seule fois dans la vie de l'équipement, voire du système. Ainsi ces précalculs ne sont pas pris en compte dans la latence d'une opération.

1.2.1 Nombres de Mersenne et Pseudo-Mersenne

La technique la plus simple et la plus efficace est l'utilisation des nombres de Mersenne, ou de leur généralisation, les nombres de pseudo-Mersenne. Ces nombres ont la particularité qu'une réduction modulaire peut être remplacée avantageusement par une ou plusieurs additions. Les nombres de Mersenne sont de la forme $p = 2^n - 1$, avec n un entier positif. Par exemple, pour un nombre de pseudo-Mersenne égal à $2^r - 1$ si un nombre s'écrit $\sum_{i=0}^{n} a_i 2^{ri}$, sa réduction modulo p sera égale à $|\sum_i (a_i)|_p$. Ainsi l'opération de réduction d'un nombre inférieur à p^2 correspond au maximum à un décalage, une addition, un test et un incrément. Les nombres de pseudo-Mersenne sont une généralisation des nombres de Mersenne. Ils concernent l'ensemble des nombres $p = 2^n - q$, avec $-\sqrt{p} < q < \sqrt{p}$. Dans ce cas, le coût de la réduction d'un tel nombre est de 3 additions modulaires, d'une multiplication $r \times log_2(q)$, et d'une multiplication $log_2(q) \times log_2(q)$ par l'algorithme 1.

On peut aisément constater que $||x_3|| \leq ||q||$, et donc que $||x_3 q|| \leq p$, par conséquent la réduction de la ligne 3 est une réduction modulaire d'un nombre compris entre $-p$ et $2^n + p$, la réduction est donc facile.

Cette technique est aisée à implémenter mais limite le choix dans les nombres premiers.

Algorithme 1 : Réduction modulaire par un nombre de pseudo-Mersenne

Entrée(s): $p = 2^n - q$, $-\sqrt{p} < q < \sqrt{p}$, $X < 2^{2n} = x_1 2^n + x_2$
Sortie(s): $|X|_p$
1: $x_3 2^n + x_4 \leftarrow x_1 q + x_2$ (avec $0 \leq x_4 < 2^n$)
2: $x_5 \leftarrow x_3 q + x_4$
3: **retourne** $|x_5|_p$

1.2.2 Réduction modulaire de Barrett

La réduction de Barrett, introduite en 1986 dans [20], permet d'obtenir un résultat approché de la valeur $|X|_p$ quand p n'a aucune caractéristique particulière et que $X < p^2$. Elle consiste en une approximation rapide du quotient X/p grâce à la valeur précalculée suivante : $\wp = \lfloor 2^{2n}/p \rfloor$, avec n tel que $p < 2^n$. Le résultat de l'algorithme de Barrett est obtenu par le calcul de l'équation suivante :

$$X' = X - \left\lfloor \left\lfloor \frac{X}{2^n} \right\rfloor \frac{\wp}{2^n} \right\rfloor p \tag{1.1}$$

Il est aisé de voir que X' est bien congru à X modulo p. De plus, Barrett [20] montre que cette valeur est inférieure à $3p$. Ainsi, il est aisé de terminer la réduction modulaire à partir de cette valeur. Le coût de cette réduction modulaire est de $1,5$ multiplications de taille n (2 multiplications de la taille de p, dont une qui ne calcule que les bits de poids fort). On peut noter que si le coût en complexité est resté le même que pour [20], des améliorations à cet algorithme ont été constamment apportées, donnant des résultats expérimentaux bien meilleurs, comme [65].

1.2.3 Montgomery

L'algorithme de réduction de Montgomery [96] n'est pas à proprement parler un algorithme de réduction, puisqu'il ne calcule pas le résultat $|X|_p$, mais $|XM^{-1}|_p$, pour une valeur M première avec p qui a les caractéristiques suivantes :

- le calcul modulo M est aisé,
- la division exacte par M est aussi aisée.

C'est notamment le cas des puissances de 2 qui proposent ces 2 opérations gratuitement (ces opérations consistent en une troncature à gauche ou à droite). L'idée de l'algorithme est de trouver la plus petite valeur positive q telle que $R = X + qp$ (qui est donc congrue à X modulo p) soit aussi congrue à 0 modulo M. Pour cela on définit la valeur précalculée $\wp = |-p^{-1}|_M$, et obtient $q = |X\wp|_M$. Comme $R < X + Mp$, et que sa division par M est une division exacte, on obtient que $S < \lceil X/M \rceil + p$. Ainsi, si $X < Mp$ alors $S < 2p$.

Algorithme 2 : Algorithme de Montgomery

Entrée(s): p,M premier avec p, X et $\wp = |-p^{-1}|_M$
Sortie(s): $|XM^{-1}|_p$
1: $Q \leftarrow |X \cdot \wp|_M$
2: $R = X + p \cdot Q$
3: $S \leftarrow R/M$
4: **retourne** S

Par ailleurs, on peut constater que dans le cas où $M < p < M^r$, cet algorithme peut être itérativement exécuté r fois pour obtenir une valeur de sortie $X' = |X/M^r|_p$, qui sera

inférieure à $2p$ pour toute entrée $X < M^r p$. Cette technique a de nombreuses variantes, une première étant donnée dans l'algorithme Simple Operand Scanning (SOS - algorithme 3), une autre dans l'algorithme Coarsely Integrated Operand Scanning (CIOS- algorithme 4), qui permet de mixer une multiplication et une réduction modulaire dans le même algorithme. Les 2 algorithmes de la figure 1.2.3 sont donnés dans le cas où $M = 2^r$. Le lecteur cherchant plus de détails sur le sujet pourra se référer à [33] qui propose d'autres variantes.

Algorithme 3 : Simple Operand Scanning

Entrée(s): $p < 2^{rn}$,
$X = \sum_i (x_i 2^{ir}) < 2^{2rn}$
Entrée(s): $\wp = |-p^{-1}|_{2^r}$
Sortie(s): $X' = |X 2^{-rn}|_p$
1: $S \leftarrow \sum_{i=0}^{n-1} (x_i 2^{ir})$
2: **pour** i de 0 à $n-1$ **faire**
3: $q \leftarrow |s_0 \cdot \wp|_{2^r}$
4: $S \leftarrow \frac{S+qp}{2^r}$
5: **fin pour**
6: **retourne** $|S|_p$ (avec $S < 3p$)

Algorithme 4 : Coarsely Integrated Operand Scanning

Entrée(s): $p < 2^{rn}$,
$X = \sum_i (x_i 2^{ir}) < p$,$Y = \sum_i (y_i 2^{ir}) < p$
Entrée(s): $\wp = |-p^{-1}|_{2^r}$
Sortie(s): $Z = |XY 2^{-rn}|_p$
1: $S \leftarrow 0$
2: **pour** i de 0 à $n-1$ **faire**
3: $Q = |(s_0 + x_0 \cdot y_i) \cdot \wp|_{2^r}$
4: $S = \frac{S+X \cdot y_i + qp}{2^r}$
5: **fin pour**
6: **retourne** $|S|_p$ (avec $S < 2p$)

Le facteur M^{-1}, s'il peut s'avérer gênant dans le cas d'une réduction isolée, ne l'est plus du tout dans le cas d'un enchaînement de calculs dans $\mathbb{Z}/p\mathbb{Z}$, comme c'est le cas dans toutes les primitives étudiées dans cette thèse. En effet, la fonction suivante :

$$\Phi \left| \begin{array}{ccc} \mathbb{Z}/p\mathbb{Z} & \rightarrow & \mathbb{Z}/p\mathbb{Z} \\ X & \rightarrow & |XM|_p \end{array} \right.$$

est un isomorphisme de l'anneau $(\mathbb{Z}/p\mathbb{Z}, +, \times)$ vers $(\mathbb{Z}/p\mathbb{Z}, +, *)$, où $*$ est une multiplication classique suivie de l'algorithme 2, l'élément neutre pour $*$ de ce nouvel anneau étant $|M|_p$. $\Phi(X)$ est le représentant de Montgomery de X. Par ailleurs on peut aussi constater que $\Phi(X) = X * |M^2|_p$, et que $\Phi^{-1} = X * 1$. Par conséquent, seuls les 2 précalculs $\wp = |-p^{-1}|_{2^r}$ et $M_2 = |2^{2rn}|_p$ sont nécessaires pour l'utilisation du représentant de Montgomery. Ces calculs doivent en revanche être réalisées en utilisant un algorithme de réduction classique (comme Newton-Raphson).

1.2.4 Algorithmes de multiplication scalaire ou d'exponentiation

L'ensemble des primitives utilisées dans cette thèse ont comme point commun d'utiliser un groupe $(\mathbb{G}, +)$, et de réaliser une multiplication d'un élément G de $\mathbb{G}$ par un scalaire k de la manière suivante :

$$k \cdot G = \underbrace{G + G + \cdots + G}_{k \text{ fois}}$$

Si le groupe $\mathbb{G}$ est noté additivement, on parle de multiplication scalaire, et s'il est noté multiplicativement, on parle d'exponentiation (k est appelé exposant).

La littérature concernant les techniques d'implémentation efficaces de cette multiplication scalaire est importante. Les plus importantes sont rappelées ci dessous. Le MSB et LSB sont deux variantes de l'algorithme appelé "square and multiply" dans le cas de

l'exponentiation ou "double and add" pour la multiplication scalaire, et qui permettent de réaliser au coût de $1,5$ opérations de groupe par bit de k en moyenne. Le MSB est moins gourmand en ressources mémoires, tandis que le LSB permet une parallélisation plus importante (l'addition et le doublement peuvent être réalisés en même temps quand le bit de l'exposant est à 1). Le sliding windows de taille w permet de limiter ce nombre d'opérations à $(w+2)/(w+1)$ par bit de k, au prix de $2^{w-1}-1$ précalculs. La présentation donnée ci-dessous ne prend pas en compte le cas où la fenêtre n'est pas complète à la fin de l'algorithme pour des raisons de simplicité. Le "Non Adjacent Form" NAF qui ne nécessite aucun précalcul mais nécessite que l'inversion dans $\mathbb{G}$ soit réalisable (ce qui est le cas pour les courbes elliptiques, mais non pour le RSA), nécessite au pire cas $1,5$ opérations par bit de k (au lieu de 2), et en moyenne $1,33$.

La liste est loin d'être exhaustive. Ainsi, on peut citer le "fixed Windows" ou méthode Yao, le "sliding Windows LSB first", les méthodes w-NAF [64] ou les méthodes de représentation hybride [9], efficaces dans le cas où le calcul de triplement est efficace dans $\mathbb{G}$ (ce qui peut être le cas pour certaines courbes elliptiques).

Algorithme 5 : Most Significant Bit

Entrée(s): $G \in \mathbb{G}$,
Entrée(s): $k \in \mathbb{N}$, $k = 1k_nk_{n-1}\cdots k_0$
Sortie(s): $k \cdot G$

1: $S \leftarrow G$
2: **pour** i de n à 0 **faire**
3: $\quad S \leftarrow S + S$
4: $\quad$ **si** $k_i = 1$ **alors**
5: $\quad\quad S \leftarrow S + G$
6: $\quad$ **fin si**
7: **fin pour**
8: **retourne** S

Algorithme 6 : Least Significant Bit

Entrée(s): $G \in \mathbb{G}$,
Entrée(s): $k \in \mathbb{N}$, $k = 1k_nk_{n-1}\cdots k_0$
Sortie(s): $k \cdot G$

1: $R \leftarrow G, S \leftarrow \mathcal{O}_{\mathbb{G}}$
2: **pour** i de 0 à n **faire**
3: $\quad$ **si** $k_i = 1$ **alors**
4: $\quad\quad S \leftarrow S + R$
5: $\quad$ **fin si**
6: $\quad R \leftarrow R + R$
7: **fin pour**
8: $S \leftarrow S + R$
9: **retourne** S

Algorithme 7 : W-Slinding Windows

Entrée(s): $G \in \mathbb{G}$,
Entrée(s): $k \in \mathbb{N}$, $k = 1k_nk_{n-1}\cdots k_0$
Sortie(s): $k \cdot G$

1: $U_0 \leftarrow 2^{w-1}G$;
2: **pour** i de 1 à $2^{w-1}-1$ **faire**
3: $\quad U_i = U_{i-1} + G$
4: **fin pour**
5: $S \leftarrow G$
6: **pour** i de n à 0 **faire**
7: $\quad S \leftarrow 2S$
8: $\quad$ **si** $k_i = 1$ **alors**
9: $\quad\quad S \leftarrow 2S$ (i fois)
10: $\quad\quad S = S + U_{k_{i-1}\cdots k_{i-w+1}}$
11: $\quad\quad i \leftarrow i - w$
12: $\quad$ **fin si**
13: **fin pour**
14: **retourne** S

Algorithme 8 : Non adjacent Form

Entrée(s): $G \in \mathbb{G}$,
Entrée(s): $k \in \mathbb{N}$
Sortie(s): $k \cdot G$

1: $k' =$ recodage NAF de k ($k' = 1k'_nk'_{n-1}\cdots k'_0$)
2: $S \leftarrow G$
3: **pour** i de 0 à n **faire**
4: $\quad S \leftarrow S + S$
5: $\quad$ **si** $k'_i = 1$ **alors**
6: $\quad\quad S \leftarrow S + R$
7: $\quad$ **sinon si** $k'_i = -1$ **alors**
8: $\quad\quad S \leftarrow S - R$
9: $\quad$ **fin si**
10: **fin pour**
11: **retourne** S

1.2.5 Le RSA et l'implémentation RSA-CRT

A titre d'exemple, je propose de donner ici une implémentation classique de la primitive de déchiffrement asymétrique RSA, appelée RSA-CRT. Celle-ci combine la multiplication modulaire et l'exponentiation.

RSA [107] est une primitive asymétrique, définie de la manière suivante : On considère (p, q) deux nombres premiers de taille équivalente, $N = p \cdot q$ appelé module RSA et e un entier appelé exposant public (en général, le même pour tout le monde). Soit $d = |e^{-1}|_{(p-1)(q-1)}$. La clé privée est le triplet (p, q, d) tandis que la clé publique est le couple (N, e). Pour toute valeur $M \in \mathbb{Z}/N\mathbb{Z}$, le chiffré $C = M^e$ a la propriété que $C^d = M$. La sécurité de RSA repose sur le fait que la connaissance de (N, e) ainsi qu'un grand nombre de couples (C, M), avec C choisi par l'attaquant ne permet pas d'obtenir (p, q, d).

Le niveau de sécurité d'une clé RSA correspond donc à la complexité du meilleur algorithme permettant de résoudre le problème ci-dessus. Moyennant certaines considérations dans le choix des éléments p et q, le meilleur algorithme est l'algorithme de factorisation "Number Field Sieve" [83] qui est sous-exponentiel en la taille de la clé. Par conséquent le niveau de sécurité d'une clé est en directe correspondance avec sa taille. Ainsi, selon [101], une clé de 1024 bits correspond à un niveau de sécurité de 80 bits, tandis qu'une clé de 3072 bits correspond à un niveau de sécurité de 128 bits.

L'implémentation RSA-CRT, quant à elle, est une implémentation de la primitive RSA quand p et q sont disponibles (soit uniquement pour la partie manipulant la clé privée de l'algorithme). Elle est donnée par l'algorithme 9. Elle utilise le théorème des restes chinois (voir section 2.2.1) pour calculer le résultat indépendamment modulo p et modulo q, puis reconstruit le résultat final. Cette méthode permet de gagner un facteur 4 sur un processeur classique, en ne nécessitant que 2 exponentiations modulaires de la moitié de la taille du module RSA, contre une exponentiation de la taille du module pour une implémentation naïve.

Algorithme 9 : Implémentation RSA-CRT

Entrée(s): p, q, d une clé RSA, C un chiffre
Entrée(s): $d_p = |d|_{p-1}$, $d_q = |d|_{q-1}$
Sortie(s): $M = |C^d|_{pq}$
1: $M_p \leftarrow |C_p^{d_p}|_p$
2: $M_q \leftarrow |C_q^{d_q}|_q$
3: $M \leftarrow M_p \cdot p + p\left(|p^{-1}|_q(M_q - M_p)\right)$
4: **retourne** M

1.3 Attaques physiques sur les composants

Le choix d'implémenter la cryptographie dans des composants possédant leur propre capacité de stockage (comme les cartes à puces) plutôt que sur n'importe quel autre support est dû au prix bien sûr, mais aussi à leur sécurité. En effet, la miniaturisation des technologies a permis de rendre extrêmement coûteuse toute tentative d'intrusion.

Néanmoins, depuis 1996 et les travaux de Kocher [86] ou de Boneh et al. [27], on sait qu'il existe des attaques ne nécessitant pas forcément des moyens lourds pour obtenir les informations secrètes. Ces attaques peuvent être classées en 2 catégories : les analyses par canaux auxiliaires et les attaques par perturbation.

Dans cette section, je présente ces différents types d'attaques en les illustrant sur RSA-CRT, la section 3.3.3 réalisera une synthèse des différentes attaques existantes sur les

courbes elliptiques.

1.3.1 Analyse de canaux auxiliaires

L'idée générale de ce type d'attaque est l'analyse de l'ensemble des signaux et grandeurs physiques émanant des calculs cryptographiques. En effet, les composants en fonctionnement consomment du courant, émettent des rayonnements électromagnétiques, le temps mis par le composant pour réaliser les calculs peut dépendre des données. Tous ces canaux d'information, dits "canaux auxiliaires", peuvent être exploités par les attaques [87, 105, 21], pour obtenir la clé.

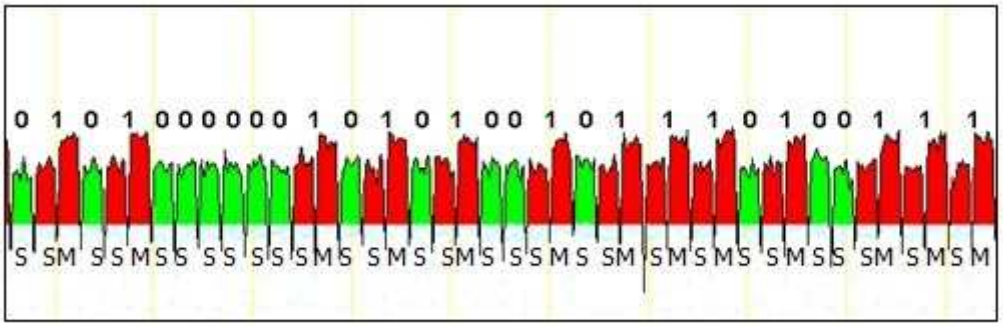
Si la cible de ce type d'attaque a initialement été la cryptographie symétrique, la cryptographie asymétrique est aussi largement concernée par la menace, et en particulier RSA et les courbes ellitiques. En effet, sur des implémentations non protégées, les attaques de type SPA [86], DPA [87] ou "template" [34] sont parfaitement réalisables.

L'attaque SPA [1] consiste à repérer dans le canal auxiliaire des informations qui seraient directement ou indirectement corrélées à la clé. tout Par exemple, la figure 1.7 permet de distinguer sur une implémentation RSA quand la valeur manipulée est mise au carré ou multipliée par une valeur différente. On peut constater que les 4 algorithmes présentés dans la section 1.2.4 sont tous sensibles à ce type d'attaque, soit directement, soit via des chaînes cachées de Markov [82] (attaque qui s'applique au DSA et seulement partiellement au RSA).

Ainsi il est important pour se protéger de ce type d'attaque de réaliser des algorithmes ne permettant pas d'obtenir de l'information sur la clé secrète simplement en distinguant l'enchaînement des opérations sur les courbes de consommation. Il existe 2 techniques : une consistant à faire converger le profil de consommation des différentes opérations (notamment sur les courbes, voir section 3.3.3), mais ce n'est pas toujours possible, l'autre étant d'utiliser un algorithme d'exponentiation (de multiplication scalaire) balancé, c'est à dire dont l'enchaînement des opérations ne dépend pas de l'exposant (du scalaire). Par exemple, des opérations inutiles ("dummy") peuvent être ajoutées, mais cela implique une vulnérabilité contre les attaques par perturbation. L'échelle de Montgomery, présentée dans l'algorithme 10, permet aussi d'atteindre cet objectif sans créer cette vulnérabilité. Il existe dans ce cas encore des attaques que l'on appelle "address bit SPA", mais qui correspondent à des attaquants très puissants capables d'obtenir la valeur des objets manipulant la clé (essentiellement les bus d'adresse) pour réaliser l'attaque.

1. SPA signifie "Simple Power Analysis", et ne considère normalement que le canal auxiliaire correspondant à la consommation instantanée du circuit. La variante "Simple Electro-Magnetic Analysis" (SEMA) traite les émanations électromagnétiques. Par simplicité, nous appelons SPA ce type d'attaque quel que soit le canal auxiliaire considéré.

FIGURE 1.7 – Une attaque SPA réussie, la clé est lue directement sur le canal de fuite



Algorithme 10 : Echelle de Montgomery et contre-mesure de Giraud [57]

ENTRÉE(S): X, d
SORTIE(S): X^d
1: $R = 1; S = X$
2: **pour** i de $log_2(d)$ à 0 **faire**
3: **si** $d_i = 0$ **alors**
4: $S = S \times R; R = R^2$
5: **sinon**
6: $R = S \times R; S = S^2$
7: **fin si**
8: **fin pour**
9: **si** $XR \neq S$ **alors**
10: détection de faute !
11: **fin si**
12: **retourne** R

Le deuxième type d'attaque, la "Differential Power Analysis", ou DPA [2] [87] correspond à un attaquant plus faible. Celui-ci aura la capacité de recueillir de nombreuses courbes et d'analyser statistiquement les courbes en fonction d'un modèle de fuite et d'une hypothèse sur une partie de la clé. L'attaquant sera alors capable de distinguer une bonne hypothèse d'une mauvaise en mesurant la corrélation [31] ou l'information mutuelle [56] entre la courbe et les informations manipulées par le composant. De nombreuses contre-mesures pour ce type d'attaque ont été proposées dans la littérature, la plus utilisée dans le cadre de la cryptographie asymétrique étant le masquage de l'opérande. Par exemple, le calcul de $|X^d|_N$ peut être réalisé en 3 étapes dans le cas du RSA (CRT ou non).

1. choix d'un aléa R et calcul de $S = R^e$ (si e est petit)
2. calcul de $T = (XS)^d$
3. retourner T/R

Ce type de contre-mesure appelé masquage multiplicatif, peut être utilisé aussi sur les courbes (voir section 3.3.3).

Enfin, le dernier type d'attaque est l'attaque par précaractérisation, ou attaque "template", où l'attaquant compare le signal issu du canal auxiliaire à une base de connaissances,

2. De même que la SPA, la DPA ne considère normalement que le canal auxiliaire correspondant à la consommation instantanée du circuit. Par simplicité, nous appelons DPA ce type d'attaque quel que soit le canal auxiliaire considéré.

et en déduit la valeur inconnue. Les attaques "template" ne nécessitent pas forcément la capacité pour l'attaquant de pouvoir injecter ses propres clés. Ainsi l'attaque de Fouque et Valette [49] généralisée par [68], permet d'obtenir cette base de connaissances sans pour autant connaître la valeur de la clé. Néanmoins la contre-mesure de masquage de l'opérande pare aussi ce type d'attaque.

1.3.2 Attaques par fautes

Les attaques par perturbations sont un autre type d'attaque permettant d'obtenir des informations sur la clé. L'objectif de ces attaques est de réaliser une perturbation et d'étudier le comportement du composant suite à cette perturbation. Les perturbations sont caractérisées par leur média d'injection : signal d'horloge, alimentation, laser, etc. Elles sont caractérisées aussi par le modèle de faute obtenu : le signal perturbé est-il un élément précis du circuit, ou la faute est-elle aléatoire et touchant plusieurs éléments en même temps ?, la perturbation est-t-elle transitoire ou permanente ?

La première attaque par perturbation publiée par Boneh [27] concerne le RSA-CRT (algorithme 9), extrêmement sensible aux perturbations. En effet, en considérant un doublet (C, M') constitué d'un chiffré et d'un résultat M' perturbé lors d'une des 2 exponentiations ou lors de la recombinaison, on obtient un des facteurs de la clé privée par le calcul suivant :

$$p \text{ ou } q = pgcd(N, M'^e - C) \tag{1.2}$$

En effet, si l'on considère M le résultat du déchiffrement de C, M est congru à M' modulo un des 2 premiers p ou q (en fonction de l'exponentiation qui aura été perturbée), et ne le sera pas modulo l'autre. Par conséquent $M'^e - M^e (= C)$ est congru à 0 modulo un des 2 premiers et pas l'autre. Le calcul du pgcd donne donc la factorisation de N.

Pour pallier cette attaque, la solution la plus simple est la vérification du calcul avec l'exposant public $C =? M^e$. Dans certains cas, cette contre-mesure est peu chère car e est un exposant assez petit (en effet, dans l'état actuel des connaissances, il n'est pas possible de remonter à M à partir de $|M^e|_N$ même pour des petites valeurs de e. Dans certains cryptosystèmes, la valeur de e est commune à tous les utilisateurs, et est généralement égale à 3, 7 ou $2^{16} + 1$). Mais dans d'autres cas, e n'est pas petit, et la vérification coûte chère (le même prix que le chiffrement). Dans certains cas, e n'est tout simplement pas disponible (par exemple, l'API RSA de javacard ne le demande pas). Par ailleurs, cette solution a le défaut de faire reposer la sécurité sur un unique test final, ce qui peut être problématique dans le cas d'un attaquant puissant capable de réaliser des perturbations assez précises.

Le développement de contre-mesures pour RSA-CRT a été un domaine de recherche assez dynamique depuis l'apparition de cette attaque, comme en témoignent les contributions [26, 36, 57, 106, 119, 39]. Ces contre-mesures peuvent être divisées en 2 familles.

La première famille utilise une extension du module, et est dérivée de l'astuce de Shamir, déposée dans le brevet [113]. Cette première proposition ayant certaines faiblesses (notamment la recombinaison n'est pas protégée) des versions plus récentes pallient ce problème et sont données dans [119, 39]. Parmi ces propositions, certaines n'utilisent pas de tests, argumentant que ceux-ci peuvent être la cible de la perturbation et renvoient la valeur perturbée, en tentant de démontrer que cette valeur ne peut pas être utilisée par l'attaquant pour retrouver la clé. Ces méthodes sont appelées méthodes "infectives", [26, 36], néanmoins à ce jour, aucune n'a pu être démontrée sûre, et même les contributions de [120, 22] ont mis en évidence un modèle de faute les mettant en défaut.

La seconde famille utilise des algorithmes d'exponentation redondants, comme par exemple l'échelle de Montgomery et variantes. La contre-mesure de Giraud [57] et Bocher [30] sont de ce type. On peut constater que la contre-mesure de Giraud ne coûte qu'une multiplication et un test si l'échelle de Montgomery est utilisée (voir algorithme 10). Rivain [106] a proposé une contre-mesure appartenant à cette famille, qui permet de réduire le nombre de multiplications et carrés de 2 par bit d'exposant pour Giraud et Bocher, à $1,65$.

Chapitre 2

Le Residue Number System : théorie et implémentation efficace

Sommaire

2.1 Introduction 27
2.2 Le Residue Number System 28
2.2.1 Le théorème des restes chinois 28
2.2.2 Application à un système de notation des nombres : le RNS . . . 29
2.3 RNS et arithmétique modulaire 30
2.3.1 Adaptation de Montgomery pour le RNS 30
2.3.2 Utilisation du Mixed Radix System 31
2.3.3 Transformations de Shenoy et Kumaresan 32
2.3.4 Approximation dans le changement de base : transformations de Posch et Kawamura 33
2.3.5 Méthode agressive : l'amélioration de Bajard et al. 34
2.4 Architecture Cox-Rower 35
2.4.1 Architecture générale du Cox-Rower 35
2.4.2 Cox 37
2.4.3 Rower 38
2.4.4 Séquencement des changements de base et besoin en mémoires et précalculs 39
2.4.5 Pilotage du Cox-Rower 40
2.5 Transformations binaire → RNS et RNS → binaire 41
2.5.1 Transformation binaire → RNS 42
2.5.2 Transformation RNS → binaire 42

2.1 Introduction

Dans cette partie, j'introduis le RNS, technique de représentation des nombres permettant une arithmétique efficace. L'idée principale est la généralisation du théorème des restes chinois pour former un système de représentation des nombres alternatif à la représentation décimale classique, ou bien la représentation en base 2, dite représentation binaire dans le reste de cette thèse. Je démontrerai dans cette partie que loin d'être une vue conceptuelle de la représentation des nombres, le RNS peut être utilisé dans une architecture dédiée de coprocesseur pour réaliser efficacement des calculs modulo un grand nombre premier ou

composé, ce coprocesseur de calculs modulaires servant de base aux primitives asymétriques telles que l'exponentiation modulaire RSA, la multiplication scalaire sur courbe elliptique ou le calcul de couplage. Je décrirai dans cette partie les points communs des 3 coprocesseurs proposés dans mes contributions [61, 35, 62], notamment concernant l'architecture globale, le séquencement et les transformations nécessaires entre la représentation binaire et la représentation RNS.

Après avoir introduit le RNS dans la section 2.2, je propose un état de l'art de l'ensemble des techniques permettant d'utiliser le RNS dans un contexte de calcul modulo un grand nombre dans la section 2.3. En particulier, je rappelle les techniques de réduction modulaire issues des travaux de Bajard *et al.* [14]. Je présente l'architecture Cox-Rower proposée par Kawamura dans [71] dans la section 2.4. Puis dans la section 2.5, j'introduis les algorithmes de transformation entre le RNS et la représentation binaire, et notamment ceux qui sont compatibles avec l'architecture de Kawamura [71]. J'y propose un nouvel algorithme ne nécessitant aucun matériel supplémentaire pour faire la transformation RNS $\rightarrow$ binaire. Cet algorithme est issu de la contribution [61]. Enfin dans la section 2.4.5, je propose un mode de séquencement du Cox-Rower basé sur un microcode, et qui permet la flexibilité de la fonction réalisée par le Cox-Rower (via une reprogrammation de ce microcode), tout en restant simple et peu consommateur en logique.

2.2 Le Residue Number System

2.2.1 Le théorème des restes chinois

Je rappelle tout d'abord le théorème des restes chinois généralisé à n nombres premiers entre eux.

Théorème 1 *Soit $(m_1; \cdots; m_n)$ un ensemble de nombres strictement positifs 2 à 2 premiers entre eux et $M = \prod_{i=1}^{n} m_i$. Soit $(x_1 < m_1; \cdots; x_n < m_n)$ des nombres positifs et $x < M$ une solution du système des n équations $x_i = |x|_{m_i}$. On a*

$$x = \left| \sum_{i=1}^{n} \left((|x_i \frac{m_i}{M}|_{m_i}) \frac{M}{m_i} \right) \right|_M \tag{2.1}$$

Il est à noter que la valeur $|m_i/M|_{m_i}$ existe uniquement parce que l'ensemble des m_i sont premiers entre eux.

Démonstration :

Il est facile de démontrer que $x = \left| \sum_{i=1}^{n} \left((|x_i \frac{m_i}{M}|_{m_i}) \frac{M}{m_i} \right) \right|_M$ est bien solution du système d'équation $|x|_{m_i} = x_i$. Il ne reste plus qu'à démontrer l'unicité de la solution.

Lemme 1 *Soit $(m_1; \cdots; m_n)$ un ensemble de nombres strictement positifs 2 à 2 premiers entre eux.*
Soit $(x, y) \in \mathbb{N}^2$. L'assertion suivante est vraie :

$$\forall i \in [1; n] \; x = |y|_{m_i} \Longrightarrow x = |y|_{\prod_{i=1}^{n} m_i}$$

Démonstration :

Afin de démontrer le lemme 1, on démontre tout d'abord sa restriction à $n = 2$ afin d'utiliser immédiatement une récurrence évidente.

Lemme 2 *Soit $(a; b)$ 2 entiers strictement positifs.*
Soit $(x; y) \in \mathbb{N}^2$.

$$|x|_a = |y|_a \text{ et } |x|_b = |y|_b \Longrightarrow x = |y|_{ab}$$

Pour démontrer le lemme 2, il suffit d'exprimer x sous la forme $x_a + ax_b + abx_{ab}$ via des divisions euclidiennes successives. On réalise la même chose avec $y = y_a + ay_b + aby_{ab}$. Par application du prémisse $|x|_a = |y|_a$, on a immédiatement $x_a = y_a$. Puis comme a et b sont premiers entre eux, on peut écrire :

$$|x|_b = |y|_b \Longleftrightarrow |(x - x_a)a^{-1}|_b = |(y - y_a)a^{-1}|_b \Longleftrightarrow |x_b|_b = |y_b|_b$$

Ce qui conclut la démonstration du lemme 2 et du lemme 1 via une récurrence évidente sur n.

Par conséquent, la solution du système d'équations $|x|_{m_i} = x_i$ est unique ce qui finit de démontrer le théorème 1.

2.2.2 Application à un système de notation des nombres : le RNS

Le théorème des restes chinois exposé ci-dessus est une idée très ancienne. Donald Knuth, dans [84], l'attribue à Sun Tsŭ au cinquième siècle. En revanche, l'idée que celui-ci peut donner un cadre efficace pour réaliser des opérations arithmétiques date des travaux de H.L. Garner de 1959 [53].

Choisissons un nombre $M = \prod_{i=1}^{n} m_i$, avec m_i premiers entre eux. On vient de démontrer que pour tout $x < M$ les résidus modulo m_i permettent de caractériser complètement x. Nous pouvons donc utiliser ces résidus comme représentation de grands nombres alternative à la représentation binaire.

Définition 1 *Une base RNS $\mathcal{B}$ est un n-uplet $(m_1; \cdots; m_n)$ d'entiers strictement positifs et 2 à 2 premiers entre eux. Nous appelons norme de $\mathcal{B}$ le nombre suivant $\mathcal{M}(\mathcal{B}) = \prod_{i=1}^{n} m_i$ et canal un élément m_i d'une base RNS.*

Définition 2 *Soit $\mathcal{B} = (m_1; \cdots; m_n)$ une base RNS et $x \in [0; \mathcal{M}(\mathcal{B})]$. Le n-uplet $(x_1; \cdots, x_n) = (|x|_{m_1}; \cdots; |x|_{m_n})$ est appelé la représentation RNS de x dans $\mathcal{B}$, et noté $(x)_{\mathcal{B}}$.*

Définition 3 *Deux bases $\mathcal{B}$ et $\mathcal{B}'$ sont premières entre elles si $\mathcal{M}(\mathcal{B})$ et $\mathcal{M}(\mathcal{B}')$ sont premiers entre eux. Dans ce cas $\mathcal{B} \cup \mathcal{B}'$ forme aussi une base RNS.*

La représentation RNS donne un cadre extrêmement efficace pour le calcul dans l'anneau $\frac{\mathbb{Z}}{\mathcal{M}(\mathcal{B})\mathbb{Z}}$, car elle permet des opérations terme à terme, sans propagation de retenue ni dépendance entre les canaux. En effet :

$$\begin{aligned}
(|X + Y|_{\mathcal{M}(\mathcal{B})})_{\mathcal{B}} &= (|x_1 + y_1|_{m_1}; \cdots; |x_n + y_n|_{m_n}) \\
(|X - Y|_{\mathcal{M}(\mathcal{B})})_{\mathcal{B}} &= (|x_1 - y_1|_{m_1}; \cdots; |x_n - y_n|_{m_n}) \\
(|X \times Y|_{\mathcal{M}(\mathcal{B})})_{\mathcal{B}} &= (|x_1 \times y_1|_{m_1}; \cdots; |x_n \times y_n|_{m_n})
\end{aligned}$$

En contrepartie, des opérations relativement simples dans la représentation binaire deviennent des algorithmes gourmands en représentation RNS. La raison essentielle est que le RNS n'est pas un système positionnel, les residus n'ayant pas de "poids". Par conséquent, la comparaison nécessite un passage dans un système de représentation positionnel, comme la représentation binaire ou la représentation MRS (voir section 2.3.2) pour être aisément exécutée. Les algorithmes de passage en représentation positionnelle étant quadratiques en n, cela rend la comparaison de 2 nombres coûteuse.

En outre la division euclidienne, et le calcul modulaire pour une valeur différente de $\mathcal{M}(\mathcal{B})$ sont aussi des opérations coûteuses (car la divison euclidenne ne peut être exécutée que dans un système positionnel). Or cette dernière est essentielle pour la cryptographie puisque sont utilisés des nombres premiers ou peu friables (comme RSA), qui par nature ne peuvent être utilisés comme bases RNS. Dans la section 2.3, nous introduisons les techniques utilisées pour pallier ce défaut.

2.3 RNS et arithmétique modulaire

2.3.1 Adaptation de Montgomery pour le RNS

Dans la section 1.2.3, j'avais présenté l'algorithme de Montgomery ainsi que ses variantes dans le contexte de la représentation binaire. Sa transposition dans le domaine de la représentation RNS a été proposée tout d'abord par [14].

L'algorithme 11 rappelle cette contribution.

Algorithme 11 : $\texttt{RedM}(X, p, \mathcal{B}, \mathcal{B}')$

Entrée(s): $\mathcal{B}$ et $\mathcal{B}'$ bases RNS premières entre elles.
Entrée(s): p premier avec $\mathcal{M}(\mathcal{B})$ et $\mathcal{M}(\mathcal{B}')$
Entrée(s): $(X)_{\mathcal{B}}$ et $(X)_{\mathcal{B}'}$ représentations RNS de X dans $\mathcal{B}$ et $\mathcal{B}'$
Entrée(s): précalculs : $(|-p^{-1}|_{\mathcal{M}}(\mathcal{B}))_{\mathcal{B}}$,$(|\mathcal{M}(\mathcal{B})^{-1}|_{\mathcal{M}(\mathcal{B}')})_{\mathcal{B}'}$ et $p_{\mathcal{B}'}$
Entrée(s): les algorithmes de changement de base $BE(A, \mathcal{B}, \mathcal{B}')$ et $BE(A, \mathcal{B}', \mathcal{B})$
Sortie(s): $(S)_{\mathcal{B}}$ et $(S)_{\mathcal{B}'}$
1: $Q \leftarrow X \times |-p^{-1}|_{\mathcal{M}(\mathcal{B})}$ dans $\mathcal{B}$
2: $Q' \leftarrow BE(Q, \mathcal{B}, \mathcal{B}')$
3: $R' \leftarrow X + Q' \times p$ dans $\mathcal{B}'$
4: $S' \leftarrow \tilde{R} \times \mathcal{M}(\mathcal{B})^{-1}$ dans $\mathcal{B}'$
5: $S \leftarrow BE(S, \mathcal{B}', \mathcal{B})$
6: **retourne** S et S'

Dans cet algorithme, $\mathcal{M}(\mathcal{B})$ joue le rôle de M présenté dans la section 1.2.3. Ainsi, il est aisé de calculer modulo $\mathcal{M}(\mathcal{B})$ à condition d'utiliser la base RNS $\mathcal{B}$. Par ailleurs la division exacte par $M(\mathcal{B})$ est elle aussi aisée, mais à condition d'utiliser une base $\mathcal{B}'$ première avec $\mathcal{B}$.

La fonction $BE(Q, \mathcal{B}_1, \mathcal{B}_2)$ nécessaire à cet algorithme est définie comme suit : pour 2 bases RNS $\mathcal{B}_1$ et $\mathcal{B}_2$, fixes, elle calcule à partir de $(X)_{\mathcal{B}_1}$ la valeur $|X|_{\mathcal{M}(\mathcal{B}_2)}$ dans $\mathcal{B}_2$. Cette fonction permet d'obtenir la représentation de X dans $\mathcal{B}_2$ à partir de la représentation dans $\mathcal{B}_1$. Ainsi la fonction BE est appelée fonction de changement de base [1] dans le reste de cette thèse.

1. Dans certaines publications, cette fonction est appelée extension de base si $pgcd(\mathcal{M}(\mathcal{B}_1), \mathcal{M}(\mathcal{B}_2)) = 1$, puisqu'elle peut être vue comme le calcul de $(X)_{\mathcal{B}_1 \cup \mathcal{B}_2}$ à partir de $(X)_{\mathcal{B}_1}$.

Théorème 2 *Soit $\alpha = \lfloor \mathcal{M}(\mathcal{B})/p \rfloor$. Si $\alpha \geq 2$, si X en entrée de l'algorithme 11 est inférieur à αp^2 et si $\mathcal{M}(\mathcal{B}')$ est supérieur à $2p$, alors S et S' sont les représentations dans $\mathcal{B}$ et $\mathcal{B}'$ d'un même nombre S, avec $S < 2p$ et $S = |X\mathcal{M}(\mathcal{B})^{-1}|_p$.*

Ce théorème est une contribution originale de [61]. Il améliore le théorème original de [14] dans la mesure où il relâche une contrainte sur les bases : $\mathcal{M}(\mathcal{B}') > \mathcal{M}(\mathcal{B})$ est remplacée par $\mathcal{M}(\mathcal{B}') > 2p$. Le paramètre α joue un rôle essentiel. En effet, si la valeur X à réduire n'est pas un simple produit, mais une somme de produits, l'utilisation d'une base plus grande permet de réduire systématiquement X en une valeur S inférieure à $3p$. Comme nous le verrons dans les chapitres 3 et 4, l'utilisation massive de réduction de sommes de produits (technique appelée réduction fainéante ou "lazy reduction") pour les algorithmes sur courbe elliptique nous obligera à calibrer cette valeur α.

Démonstration :

Nous considérons $X < \alpha p^2$, $\mathcal{M}(\mathcal{B}) > \alpha p$ et $\mathcal{M}(\mathcal{B}') > 2p$. Soit $q = |-Xp^{-1}|_{\mathcal{M}(\mathcal{B})}$. Par définition de q, nous avons aisément $|X + qp|_{\mathcal{M}(\mathcal{B})} = 0$. Par ailleurs $X + qp < 2p\mathcal{M}(\mathcal{B})$. Nous considérons la valeur $s = (X + qp)/\mathcal{M}(\mathcal{B})$. Nous avons bien $s < 2p$ ainsi que $s = |X\mathcal{M}(\mathcal{B})^{-1}|_p$.

Démontrons maintenant que l'algorithme 11 revient à calculer le représentant de s dans les 2 bases $\mathcal{B}$ et $\mathcal{B}'$. De manière évidente nous avons $Q = (q)_{\mathcal{B}}$. Par conséquent suite à la ligne 2, $Q' = (|q|_{\mathcal{M}(\mathcal{B}')})_{\mathcal{B}'}$.

Comme $(X)_{\mathcal{B}'} = (|X|_{\mathcal{M}(\mathcal{B}')})_{\mathcal{B}'}$ et $p < \mathcal{M}(\mathcal{B}')$, nous avons l'égalité suivante :

$$S' = \left(\left| \frac{X + qp}{\mathcal{M}(\mathcal{B})} \right|_{\mathcal{M}(\mathcal{B}')} \right)_{\mathcal{B}'} = \left(|s|_{\mathcal{M}(\mathcal{B}')} \right)_{\mathcal{B}'} = (s)_{\mathcal{B}'}$$

Enfin comme $\alpha > 2$ $S = (s)_{\mathcal{B}}$.

Comme nous pouvons le constater, jusqu'à présent l'algorithme 11 permet les traitements indépendamment sur chaque canal de la base $\mathcal{B} \cup \mathcal{B}'$ et donc un traitement facilement parallélisable dans une architecture dédiée. En revanche la fonction $BE(X, \mathcal{B}, \mathcal{B}')$ est un traitement traditionnellement consommateur en temps de calcul. Il existe dans la littérature plusieurs manières de réaliser ce traitement, que je propose de détailler ci dessous.

2.3.2 Utilisation du Mixed Radix System

Naturellement, l'algorithme le plus naïf pour réaliser le changement de base est le passage par la représentation binaire à partir de la représentation dans $\mathcal{B}_1$, puis une retransformation dans la base $\mathcal{B}_2$. Si la complexité (comptée en nombre de multiplications élémentaires sur un chiffre) de cette fonction est la même que tous les algorithmes exposés dans cette partie ($O(n^2)$ dans le cas d'un choix de base aléatoire), cet algorithme réintroduit l'arithmétique binaire, et donc le passage des retenues, nous perdons ainsi l'intérêt du RNS par rapport à l'approche classique. Il est à noter que la méthode de passage à la représentation classique a été utilisée par [110] pour réaliser un coprocesseur pour multiplication modulaire et courbes elliptiques basé sur le RNS, et les résultats obtenus, bien inférieurs à ceux que j'ai obtenus dans [61] confortent l'idée que cette solution est à éviter.

Le premier algorithme de la littérature permettant le changement de base sans passage par la représentation binaire est reporté dans un papier de 1969 dû à Szabo et Tanaka

Algorithme 12 : RNS2$MRS(X, \mathcal{B}_1)$

Entrée(s): $\mathcal{B} = (m_1, \cdots, m_n)$ une base RNS.
Entrée(s): $\mu_{i,j} = |{m_i}^{-1}|_{m_j}$ pour $i < j$
Entrée(s): $(x_1, \cdots, x_n) = (X)_{\mathcal{B}}$
Sortie(s): la représentation MRS de X dans la base $(m_1, \cdots, m_n)$

1: $x'_1 \leftarrow x_1$
2: $x'_2 \leftarrow |(x_2 - x'_1)\mu_{1,2}|_{m_2}$
3: $x'_3 \leftarrow |((x_3 - x'_1)\mu_{1,3} - x'_2)\mu_{2,3}|_{m_3}$
4: $\cdots$
5: $x'_n \leftarrow |(\cdots((x_n - x'_1)\mu_{1,n} - x'_2)\mu_{2,n})\cdots - x'_{n-1})\mu_{n-1,n}|_{m_n}$
6: **retourne** $(x'_1, \cdots x'_n)$

[116]. Il utilise pour cela la représentation Mixed Radix System où MRS, définie comme suit :

Définition 4 *Soient $(m_1 \cdots m_n)$ n entiers strictement positifs. Considérons $m_0 = 1$. Soit $X < \prod_{i=1}^{n} m_i$. La représentation de X dans la base MRS $(m_1 \cdots m_n)$ est l'unique n-uplet $x'_1, \cdots, x'_n$ tel que :*

- $\forall i \in [1; n] x'_i < m_i$
- $X = \sum_{i=1}^{n}(x'_i \prod_{j=0}^{i-1} m_j)$

Autrement dit, la représentation MRS est une généralisation de la représentation classique, la base MRS de la représentation classique étant composé d'une unique valeur répétée n fois (10 pour le système décimal, 2 pour la représentation binaire). Szabo et Tanaka [116] proposent d'utiliser la base MRS constituée des éléments de la base RNS. L'algorithme 12 propose une version de cette transformation.

Il est à noter que cet algorithme nécessite les valeurs précalculées suivantes $\mu_{i,j} = |{m_j}^{-1}|_{m_i}$ pour $i > j$, soit $n(n-1)/2$ précalculs. Dans [13], les auteurs proposent une variante où le nombre de précalculs peut tomber à n au prix d'une séquentialisation de l'algorithme (chaque opération est tributaire de l'opération antérieure).

Une fois la valeur de X en représentation MRS obtenue, la transformation RNS dans l'autre base se fait par l'algorithme naïf.

Si l'on considère la complexité comme étant le nombre de multiplications modulo les m_i pour l'algorithme de changement de base, on remarque que la complexité $\mathcal{C}(BE_{\mathrm{MRS}}) = 3/2n^2 + o(n^2)$ pour une base aléatoire. Néanmoins, on peut noter que pour chaque multiplication, une des opérandes est un élément ne dépendant que de la base, qui peut être précalculée. De plus, il est possible de réaliser beaucoup mieux en choisissant les bases de telle manière que les valeurs précalculées soient des petits nombres ou des puissances de 2, afin de remplacer les multiplications par des additions. Cette étude, reportée dans [17], permet de limiter la complexité de BE à $7/10n^2 + o(n^2)$.

Néanmoins, cet algorithme a le défaut d'être difficilement parallélisable dans une architecture dédiée, en grande partie à cause de la transformation RNS vers MRS, qui présente beaucoup de dépendances entre les résultats intermédiaires.

2.3.3 Transformations de Shenoy et Kumaresan

Dans l'article de Shenoy et Kumaresan de 1989 [114], les auteurs proposent un nouvel algorithme pour réaliser le changement de base BE_{SK}. L'idée de l'algorithme est de réécrire

l'équation 2.1 sous la forme suivante :

$$x = \left(\sum_{i=1}^{n} \xi_i \frac{\mathcal{M}(\mathcal{B}_1)}{m_i} \right) - \lambda \mathcal{M}(\mathcal{B}_1) \text{ ou } \xi_i = \left| x_i \left(\frac{\mathcal{M}(\mathcal{B}_1)}{m_i} \right)^{-1} \right|_{m_i} \tag{2.2}$$

L'évaluation du premier terme de la soustraction peut être réalisée dans tous les canaux de $\mathcal{B}_2$, avec un coût de n^2 multiplications. La difficulté est de pouvoir évaluer λ. Pour cela, Shenoy et Kumaresan proposent l'utilisation d'un élément supplémentaire m_χ premier avec les canaux de la base $\mathcal{B}_1$ et la base $\mathcal{B}_2$ et supérieur à n (qui est la valeur maximale de λ. Pour réaliser cet algorithme, il est nécessaire d'avoir la valeur de $x_\chi = |X|_{m_\chi}$ en début d'algorithme, ce qui rend cette méthode inutilisable pour le premier changement de base de l'algorithme 11.

La valeur λ est ainsi obtenue par la formule suivante :

$$\lambda = \left| \left(\sum_{i=1}^{n} \xi_i \frac{\mathcal{M}(\mathcal{B}_1)}{m_i} - x_\chi \right) .\mathcal{M}(\mathcal{B}_1)^{-1} \right|_{m_\chi} \tag{2.3}$$

La complexité de Shenoy et Kumaresan est de $n(n+1)$ multiplications modulaires, ajouté à $2n+1$ multiplications par un opérande de taille $(log_2(n))$ pour le calcul de λ. Aucune contrainte n'est requise sur le choix des bases via cette méthode.

2.3.4 Approximation dans le changement de base : transformations de Posch et Kawamura

Dans l'article [104] Posch et Posch montrent que la valeur λ de l'équation 2.2 est en fait :

$$\lambda = \lfloor \left(\sum_{i=1}^{n} \frac{\xi_i}{m_i} \right) \rfloor \tag{2.4}$$

Comme il s'agit d'une somme de fractions, il n'est pas possible de calculer exactement λ, il est nécessaire de réaliser une approximation, avec le risque de voir apparaître des erreurs de calculs, ce qui est extrêmement dangereux quand on réalise de la cryptographie (voir chapitre 1.3.2). Dans l'article [71], Kawamura réalise le tour de force de supprimer les effets des mauvaises approximations dans les 2 changements de base de la multiplication de Montgomery.

Pour ce faire, il choisit les canaux constitutifs de $\mathcal{B}$ et $\mathcal{B}'$ comme des pseudo-Mersenne de même taille, soit des éléments $m_i = 2^r - \varepsilon_i$. Ainsi l'auteur peut approcher la valeur $eval(\xi_i, m_i)$ par $trunc_l(\xi_i/2^l)$, où la fonction $trunc_l$ est définie comme suit : $trunc_l(x) = \lfloor x/2^{(r-l)} \rfloor$ (soit la fonction qui garde uniquement les l bits de poids fort). Par développement limité et dans la mesure où $\xi_i < m_i$, on peut constater facilement que

$$-2^{-l} - 2^{-r/2+1} < approx_i = trunc_l(\xi_i) - \frac{\xi_i}{m_i} \leq 0$$

Par conséquent

$$-errmax = (-2^{-l} - 2^{-r/2+1})n < \sum_{i=1}^{n} approx_i \leq 0$$

Par conséquent, lors d'un changement de base, la valeur λ sera estimée par λ' qui sera égale à λ ou $\lambda - 1$.

Lors du premier changement de base, si $\lambda' = \lambda - 1$ la valeur calculée de $\tilde{S'}_{\mathcal{B}'}$ à la ligne 4 de l'algorithme 11 correspond à $S' + p_{\mathcal{B}'}$, ce qui n'a pas d'autres conséquences

pour l'algorithme 11 que de fournir un résultat compris entre 0 et $3p$. La seule condition nécessaire est que $\mathcal{M}(\mathcal{B}') > 3p$ afin que $\tilde{S}'$ puisse être représenté dans $\mathcal{B}'$.

Lors du deuxième changement de base, si $\lambda' = \lambda - 1$, alors le résultat de la ligne 5 est $\tilde{S} = (S + \mathcal{M}(\mathcal{B}'))_{\mathcal{B}}$, ce qui n'est évidemment pas acceptable. Pour pallier ce problème, Kawamura introduit une valeur $errinit$ comprise entre $errmax$ et 1 non inclus, et augmente la taille de $\mathcal{B}'$ de telle manière que :

$$\mathcal{M}(\mathcal{B}') > \frac{3p}{1 - errinit} \tag{2.5}$$

Ensuite λ est estimée par $\lambda'' = \lambda' + errinit$. Grâce à cet ajout, nous savons que

$$\lambda \leq \lambda'' \leq \lambda + errinit$$

Or

$$\lambda - \lfloor \lambda \rfloor = \frac{X}{\mathcal{M}\mathcal{B}'} \leq \frac{3p}{\mathcal{M}(\mathcal{B}')}$$

Donc

$$\lambda - \lfloor \lambda \rfloor < 1 - errinit$$

Ce qui permet de conclure :

$$\lfloor \lambda'' \rfloor = \lfloor \lambda \rfloor$$

Ainsi aucune mauvaise approximation ne peut être réalisée lors de ce deuxième changement de base.

Kawamura a donc réussi à remplacer l'approximation de λ par des simples additions sur un vecteur de taille l. Dans [99], les mêmes auteurs ont par ailleurs démontré que pour $r = 32$, $l = 8$ et $errinit = 1/2$ sont largement suffisants pour réaliser du RSA 2048 (soit $n = 65$). Si l'on exclut les additions, la complexité du changement de base de Kawamura et al. [71] tombe ainsi $n(n + 1)$. Comme nous le verrons, cet algorithme est aussi aisément parallélisable, ce qui le rend plus intéressant que l'utilisation du MRS si l'on souhaite réaliser une architecture matérielle. L'algorithme 13 décrit le changement de base de Kawamura.

2.3.5 Méthode agressive : l'amélioration de Bajard et al.

Suivant les idées de Kawamura et al., Bajard et al. [15] proposent la suppression complète du calcul de δ, dans la mesure où cela n'affecte pas le résultat final modulo p. Cela a néanmoins un impact sur la deuxième base, puisque la sortie s de l'algorithme 11 devient comprise entre 0 et $(n + 1)p$

Cette méthode est particulièrement efficace sur les architectures de machines classiques telles que les CPU ou les GPU [117] : en effet, ces machines ont une architecture qui est en général un diviseur de la taille des problèmes cryptographiques rencontrés (par exemple, les microprocesseurs standards sont basés sur des architectures 32 bits, tandis que les tailles classiques du RSA sont 1024, 2048, 4096 et celles des courbes elliptiques de 160 ou 256 bits). Afin de profiter du rendement maximum de chaque opération, il est préférable de prendre r égal à la taille de ce mot machine. Par conséquent, dans la mesure où chaque valeur est comprise entre 0 et lp (où l dépend du choix réalisé pour les changements de base mais est nécessairement supérieur à 2), un mot machine supplémentaire est systématiquement nécessaire. Dans ce cas, le facteur $n + 1$ n'est pas un problème vue la taille des problèmes considérés, et la méthode agressive est extrêmement compétitive.

Algorithme 13 : Algorithme de changement de base de Kawamura

Entrée(s): $\mathcal{B}_1 = (\mu, \cdots m_n)$ et $\mathcal{B}_2 = (m'_1, \cdots m'_n)$ deux bases RNS premières entre elles
Entrée(s): $(X)_{\mathcal{B}_1} = (x_1, \cdots, x_n)$
Sortie(s): $BE(X, \mathcal{B}_1, \mathcal{B}_2)$

pour i de 0 à n **faire**
  $\xi_i = |x_i m_i / \mathcal{M}(\mathcal{B}_1)|_{m_i}$
fin pour
$\delta = errinit$
pour i de 0 à n **faire**
  $r_i = 0.$
fin pour
pour i de 0 à n **faire**
  $\delta = \delta + eval(\xi_i, m_i)$
  pour j de 0 à n' **faire**
    $r_j = |r_j + \xi_i \mathcal{M}(\mathcal{B}_1)/m_i|_{m'_j}$
    si $\delta >= 1$ **alors**
      $r_j = |r_j - \mathcal{M}(\mathcal{B}_1)|_{m'_j}$
    fin si
  fin pour
  $\delta = \delta - \lfloor \delta \rfloor$
fin pour
retourne $(r_1, \cdots, r_n)$

En revanche, les FPGA ne définissent pas la taille du mot machine, même si comme nous l'avons vu dans la section 1.1.3 il est largement préférable de maintenir cette taille r en-dessous de 36 bits (à cause de la taille des multiplieurs). On peut donc tailler les bases $\mathcal{B}$ et $\mathcal{B}'$ au plus juste en utilisant $r = 33$, l'impact de l'optimisation de Bajard peut s'avérer non négligeable sur r (et donc sur la taille du cicuit et la fréquence maximale de l'horloge).

2.4 Architecture Cox-Rower

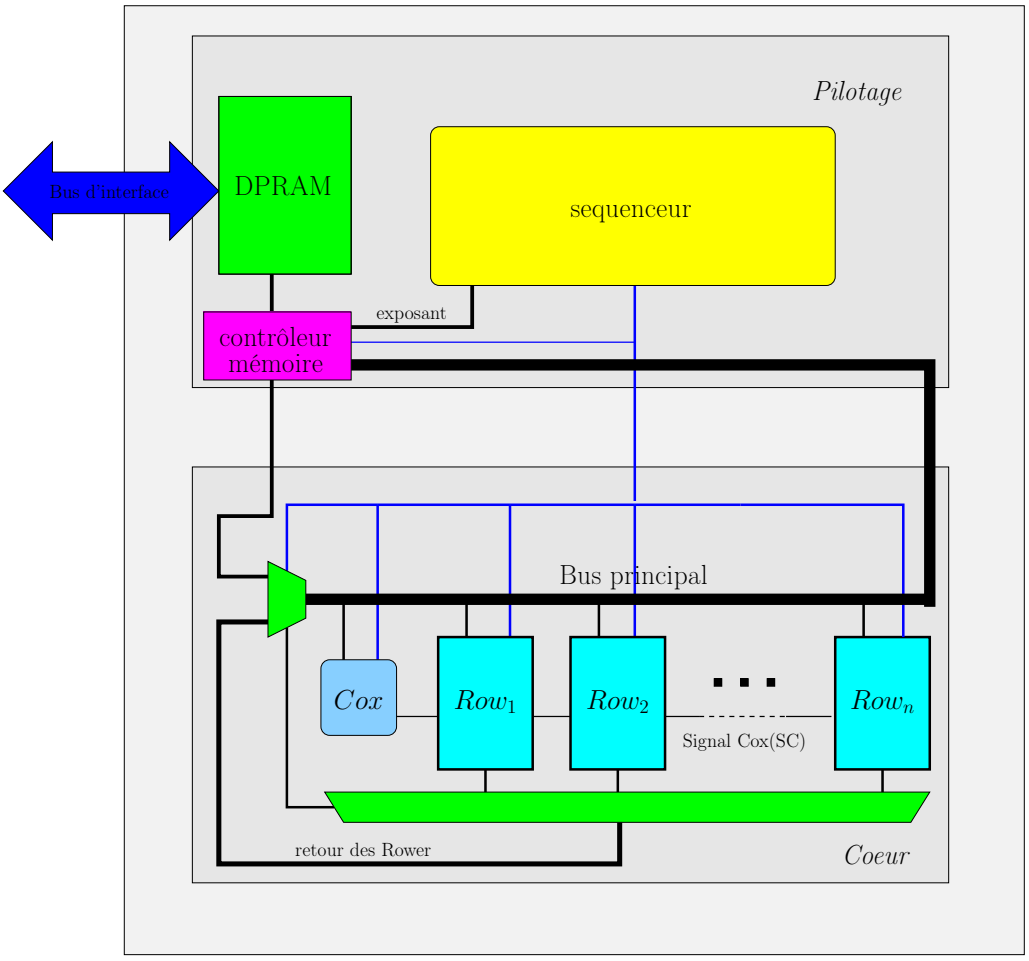
Dans cette section, j'introduis l'architecture proposée par Kawamura dans [71] et baptisée Cox-Rower. L'ensemble des éléments de cette architecture permettent de réaliser l'algorithme de réduction (algorithme 11) de manière extrêmement efficace.

L'architecture de Kawamura a été implémentée dans chacune de mes contributions [61, 35, 62], mais a fait l'objet d'un travail spécifique en fonction de l'objectif assigné aux coprocesseurs (courbes elliptiques, couplage ou contre-mesure contre les canaux auxiliaires et les fautes). Dans cette section, je présente les points communs de ces 3 architectures. Les sections 2.4.2 et 2.4.3 présentent la contribution de Kawamura [71], puis dans les sections suivantes, j'introduirai des contributions originales à cette thèse, comme la gestion de la mémoire dans la section 2.4.4 ou le pilotage du Cox-Rower dans la section 2.4.5.

2.4.1 Architecture générale du Cox-Rower

L'architecture Cox-Rower proposée par [71] a été aujourd'hui implémentée dans l'ensemble des contributions de la littérature traitant le problème de l'implémentation matérielle efficace par utilisation du RNS, à l'exception de Schinianakis et al. [110]. En effet, cette architecture s'avère être extrêmement efficace et suffisamment flexible pour réaliser

FIGURE 2.1 – Architecture générale du Cox-Rower



l'arithmétique modulaire nécessaire aux primitives asymétriques classiques (RSA et courbes elliptiques sur corps de grande caractéristique). Par ailleurs, le parallélisme de cette architecture s'adapte parfaitement à l'architecture des FPGA récents, qui embarquent de plus en plus de multiplieurs à des fins de traitement du signal.

Il est à noter que les figures de cette section n'intègrent pas les barres de registres nécessaires aux étages de pipeline. En effet, le positionnement de ces barres de registres doit être traité au cas par cas en fonction de la technologie considérée et du problème considéré. De leur bon positionnement dépend en effet la fréquence d'horloge maximale du circuit (comme nous l'avons présenté dans la section 1.1).

La figure 2.1 fournit une schématique simplifiée de l'architecture. Il s'agit d'un circuit cadencé par une unique horloge. Les entrées sorties sont échangées via une mémoire double port avec une interface de lecture sur un bus système (en bleu). En effet, s'il est possible d'utiliser le Cox-Rower seul, il est largement préférable de l'inclure dans un composant de type "System on Chip", où le coprocesseur sera piloté par un CPU principal de faible puissance, qui cadencera l'ensemble du système. Le Cox-Rower permettra ainsi d'accélérer les traitements nécessitant de l'arithmétique des grands nombres.

L'architecture du Cox-Rower est séparée en 2 : la partie supérieure réalise le pilotage

du module. Cette partie cadence les traitements du cœur efficacement (voir section 2.4.5). L'ensemble du flux de contrôle est représenté en bleu sur le schéma 2.1 ainsi que dans tous les schémas de cette thèse. La partie inférieure, le cœur du coprocesseur, est en charge de réaliser les traitements sur les données

Le cœur présenté ici est composé des 3 éléments suivants :

- un Cox, en charge du calcul de λ
- ν Rowers dont l'objectif principal est la réalisation de l'ensemble des calculs dans un canal. La nature des algorithmes 11 et 13 rend nécessaire pour une efficacité optimale que ν divise n, sans quoi certains Rowers seraient inoccupés certains cycles pendant le traitement. Le rapport $\eta = n/\nu$ correspond alors au nombre de canaux de chaque base traités par le même Rower. En particulier, dans l'ensemble de mes contributions, je privilégie $\nu = n$, afin d'obtenir les performances en délai les plus optimales. Ainsi chaque Rower a pour rôle de réaliser les calculs sur 2 canaux : un appartenant à $\mathcal{B}$, l'autre appartenant à $\mathcal{B}'$. Un rapport η différent peut se justifier pour des raisons d'économie de portes logiques (la contribution [99] propose ainsi un rapport 3 pour une implémentation RSA). Il faut noter cependant que la plupart des primitives cryptographiques nécessitent des variables temporaires qui ne sont pas compressibles. La réduction par exemple par 2 des Rowers permet ainsi d'économiser environ la moitié de la logique, mais les mémoires nécessaires sont alors 2 fois plus grandes dans chaque Rower. Ainsi, le meilleur compromis temps surface est obtenu pour $n/\nu = 1$.
- Un système de communication basé sur un bus principal, en charge de transmettre les entrées/sorties au cœur, en charge aussi de réaliser le transfert entre les Rowers ou des Rowers vers les entrées/sorties. Cette capacité est nécessaire lors du changement de base de Kawamura, où l'ensemble des valeurs ξ_i calculées à la ligne 2 de l'algorithme 13 sont nécessairement passées à l'ensemble des Rowers.

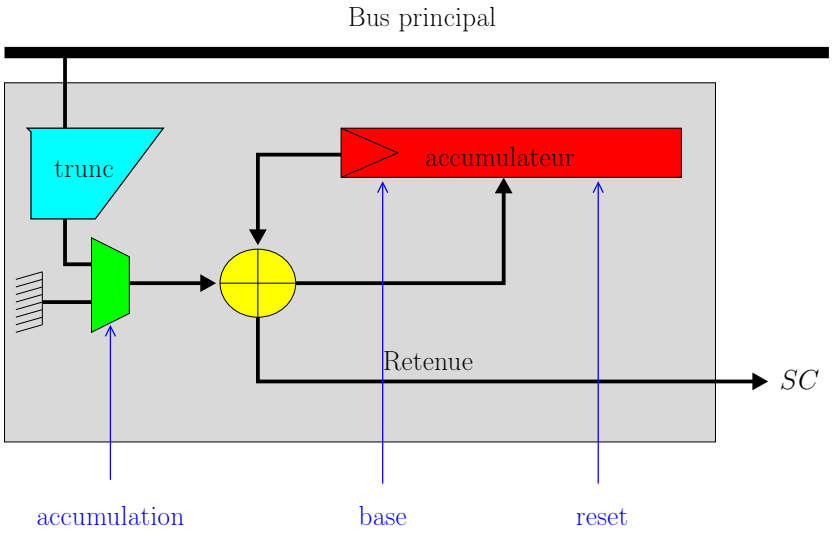
2.4.2 Cox

L'objectif du Cox est le calcul de la valeur λ lors du changement de base. Par conséquent son architecture est extrêmement simple et décrite dans la figure 2.2. L'unique sortie du Cox est la génération du *Signal Cox*, qui déclenche l'exécution de la ligne 13 de l'algorithme 13 au sein de chaque Rower. Ce bit n'est en fait que le signal de retenue du bit de poids fort de l'additionneur. Seuls 3 bits pilotent donc l'accumulateur :

- le signal de reset, qui indique que le registre d'accumulation doit se réinitialiser.
- le signal de choix de base, qui pilote la valeur de l'initialisation du registre : en effet en fonction du changement de base à réaliser (de $\mathcal{B}$ vers $\mathcal{B}'$ ou de $\mathcal{B}'$ vers $\mathcal{B}$), la valeur de l'initialisation de ce registre est respectivement 0 ou *errinit* telle que définie dans l'algorithme 13.
- le signal d'accumulation, qui pilote une porte "et" entre le bus principal et l'additionneur, permettant ainsi l'accumulation pendant les changements de base

En dehors des changements de base, le Cox est complètement inactif (le bit de contrôle *accumulation* reste à 0). Les auteurs de [99] ont proposé, vue la simplicité du Cox, de doter chaque Rower d'un Cox. L'intérêt évident est de limiter la charge (ou "fan out") du signal *Sortie Cox*. Néanmoins lors des travaux que nous avons réalisés, ce signal ne figurait jamais sur le chemin critique du circuit, ce qui rendait ce type d'optimisation inutile.

FIGURE 2.2 – Le Cox : un accumulateur à démarrage piloté



2.4.3 Rower

Le Rower est l'élément central de l'architecture. Comme indiqué ci-dessus, chaque Rower est en charge d'exécuter les traitements de η canaux sur chaque base. A l'exception du changement de base, chaque valeur calculée sur un canal n'a pas vocation à être utilisée dans un autre canal. Par conséquent, il est naturel que la gestion des variables temporaires et des précalculs nécessaires à l'ensemble des primitives que nous rencontrerons dans cette thèse soit confiée au Rower, et non gérée de manière globale, comme c'est par exemple le cas dans les applications binaire comme [108].

Chaque Rower est ainsi composé de 3 parties distinctes que l'on peut voir sur la figure 2.3 :

- une unité de traitement arithmétique,
- une unité de gestion de la mémoire et des précalculs,
- une unité de gestion des sorties

L'unité de traitement arithmétique : comme nous le verrons dans cette thèse, il s'agit de l'unité traitant le flot de données. Celle-ci peut être spécifique au traitement réalisé, tant au niveau du nombre d'étages de pipeline que de la fonctionnalité même qu'elle est capable de réaliser.

Il s'agit dans tous les cas d'un automate synchrone à mémoire, dont la sortie est égale à l'état interne, noté $U(t)$. Afin de pouvoir réaliser efficacement les traitements minimaux requis par l'ensemble des algorithmes cryptographiques, $U(t)$ doit être en mesure de calculer la fonction suivante :

$$U(t)(X_0, X_1, C, \varepsilon, \alpha, \beta) = |X_0 \times X_1 + \alpha U(t-1) + \beta C|_{2^r - \varepsilon} \tag{2.6}$$

où X_0, X_1 et C sont des entiers positifs compris entre 0 et 2^r, α égal à 0 ou 1, et ε un entier positif de petite taille.

La nature de cette unité de traitement arithmétique permet ainsi de traiter efficacement tout type de calcul rencontré en cryptographie. En effet, cette unité permet le

traitement d'une multiplication complète en 2η cycles (η cycles sur chaque base RNS). L'addition est quant à elle traitée en 4η cycles, car elle nécessite l'utilisation de l'accumulateur. Néanmoins, elle peut être combinée avec des multiplications sans ajout de cycles supplémentaires, pour traiter des calculs de la forme $X_0X_1 + X_2X_3 + \cdots + X_{l-1}X_l$, qui sont typiques des schémas de réduction fainéante ou "lazy reduction" (voir section 3.2.4).

L'objectif de C et l'intérêt d'une mémoire spécifique associée dans la figure 2.3 sont expliqués dans la section 2.4.4. β n'est ainsi pas un signal piloté par le séquenceur, mais le signal issu du Cox appelé SC sur les figures 2.2 et 2.3.

Les mémoires : l'ensemble des modules de stockage (en rouge dans la figure 2.3) est en charge de stocker les précalculs ainsi que les variables temporaires du Rower. Ils sont au nombre de 3.

- La mémoire principale est en charge de stocker les valeurs X_0 et X_1 à chaque cycle, ainsi que de charger le résultat de l'ALU. La manière de l'implémenter la plus simple est donc l'utilisation d'une RAM 3-ports (2 en lecture et 1 en écriture). Néanmoins vu le coût de ce type de RAM dans des technologies différentes de celles des FPGA, il est possible de structurer ces mémoires différemment comme nous le verrons dans le chapitre 3 où j'ai fait le choix de n'utiliser aucune mémoire RAM embarquée.
- La mémoire ROM_1 est en charge du stockage des éléments de la base. Elle contient donc 2η éléments fixes de petite taille (inférieur à $r/2$). L'utilisation d'une mémoire structurée pour une taille si faible (rappelons que $\eta = 1$ est à privilégier) n'est pas nécessaire, elle peut être réalisée via de la logique fixe (ROM) ou des registres en fonction du besoin de réaliser des changements dans les valeurs contenues.
- La mémoire ROM_2 doit permettre de fournir la valeur C et est pilotée par le signal Cox, pour le traitement de la ligne 13 dans l'algorithme 13. Celle-ci ne contient par ailleurs que $\eta(\pi + 1)$ éléments, où π est le nombre de valeurs modulaires qui sont traitées en même temps (en général 1 pour les courbes elliptiques, la caractéristique du corps, et 2 pour le RSA-CRT, p et q. Elle aussi peut être traitée sous forme de ROM.

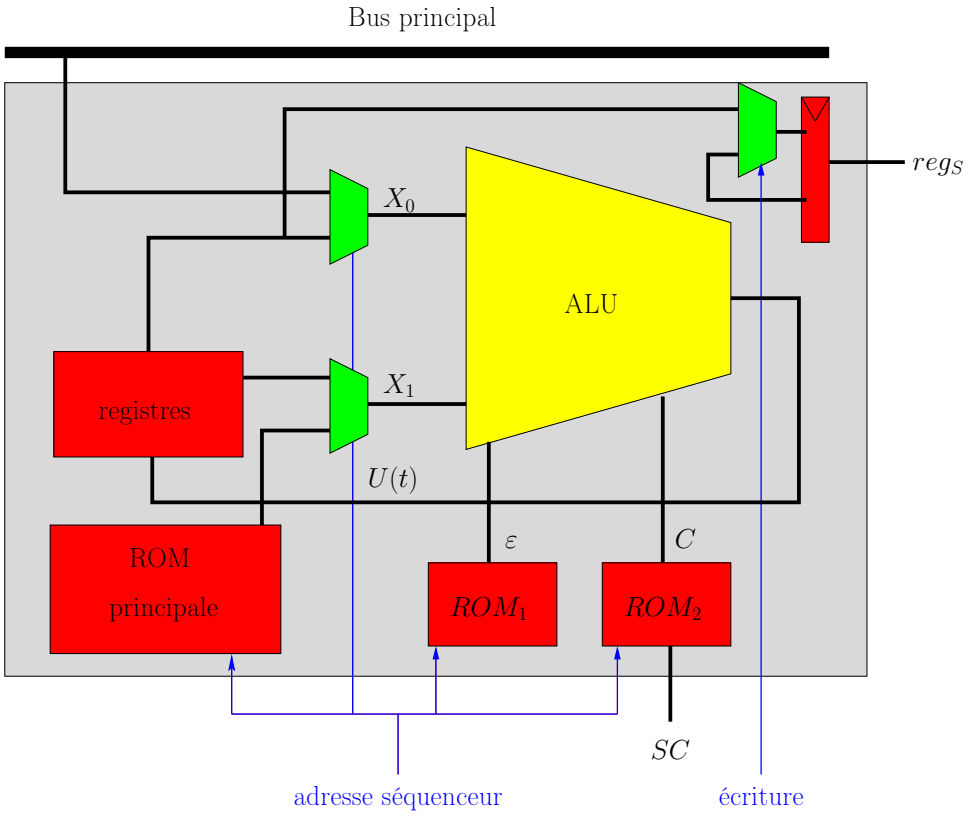
Le registre Reg_S : Le registre Reg_S a pour rôle de réaliser la sortie du Rower vers le reste du circuit. L'écriture de ce registre est pilotée par le séquenceur, tandis que la valeur écriture est mutualisée avec la sortie X_0 de la mémoire principale. L'intérêt principal de mutualiser l'écriture de ce registre avec X_0 et non avec X_1 est qu'il permet un accès à la mémoire principale vers Reg_S simultanément avec le traitement par l'ALU d'une valeur sur le bus principal avec la donnée X_1. Cela permet ainsi de traiter séquentiellement plusieurs changements de base sans perdre de cycle à l'écriture dans ce registre.

2.4.4 Séquencement des changements de base et besoin en mémoires et précalculs

La nature de l'unité de traitement permet via l'algorithme 14 de traiter l'intégralité d'une réduction de Montgomery en $\eta(2n + 3)$ cycles seulement, et ce quelle que soit la valeur de p ou de la base RNS.

L'algorithme 14 démontre l'intérêt du traitement de la ligne 13 de l'algorithme 13 en parallèle du reste des traitements, et donc de la mémoire ROM_2. Cela permet d'économiser 2η cycles sur chaque réduction. Nous verrons dans les chapitres suivants que ce dispositif s'intègre parfaitement dans les structures pipelinées des ALU que nous avons dessinées. Il faut noter que c'était déjà le cas dans la proposition de Nozaki et al [71].

FIGURE 2.3 – Le Rower : unité de traitement et de stockage des variables temporaires



Dans l'algorithme 14 sont notées les attentes pipeline. En effet, la nature de la fonction 2.6 ne permet pas d'espérer le résultat en un cycle. Le traitement de cette fonction dans une ALU pipelinée permet d'augmenter fortement la fréquence du Cox-Rower, en revanche elle induit des effets d'attente. Ces temps d'attente doivent ainsi être remplis par des calculs indépendants. Un effort de parallélisation est par conséquent nécessaire et dépend du traitement considéré. Dans les chapitres suivants, nous traiterons ainsi la parallélisation des algorithmes, pour atteindre un taux d'occupation du pipeline proche du maximum.

2.4.5 Pilotage du Cox-Rower

La structure inhérente du Cox-Rower et des calculs à réaliser sur celui-ci permet un pilotage extrêmement simplifié. En effet, dans l'ensemble des algorithmes étudiés dans cette thèse, l'ensemble des Rower peuvent exécuter la même instruction, en opérant des données spécifiques se trouvant dans ses mémoires. A ce titre, nous sommes très proche d'une architecture "Simple Instruction Multiple Data"(SIMD), que l'on retrouve classiquement dans les coprocesseurs de traitement du signal, comme les GPU.

Ainsi le choix a été fait de développer un séquenceur microcodé, dont les capacités sont spécifiques aux opérations cryptographiques traitées dans cette thèse. En effet, le flot de contrôle des primitives cryptographiques étudiées ne dépend jamais des données manipulées dans le Cox-Rower, mais uniquement d'une seule valeur, l'exposant (que l'on retrouve dans les algorithmes de la section 1.2.4 ou dans l'algorithme 10). L'exposant est donc lu dans un registre à décalage Reg_e, et les capacités de branchement du séquenceur ne dépendent que

Algorithme 14 : Algorithme de réduction de Montgomery en $\eta(2n+3)$ cycles sur un Cox-Rower de rapport η. Chaque opération est exécutée sur chacun des Rower sur η cycles pour le traitement sur la base complète.

Entrée(s): $X = (x_1, \cdots x_n, x'_1, \cdots, x'_n)$ dans $\mathcal{B}$ et $\mathcal{B}'$
Entrée(s): $P_{i,\mathcal{B}}, P_{i,\mathcal{B}'}, Q_{i,\mathcal{B}'}, M_{i,j,\mathcal{B}\to\mathcal{B}'}, M_{i,j,\mathcal{B}'\to\mathcal{B}}, C_{i,\mathcal{B}\to\mathcal{B}'}$ et $C_{i,\mathcal{B}'\to\mathcal{B}}$ précalulés (voir le tableau 2.1).
Sortie(s): $S = |X\mathcal{M}(\mathcal{B})^{-1}|_p$ dans $\mathcal{B}$ et $\mathcal{B}'$
1: $\xi_{i,\mathcal{B}\to\mathcal{B}'} = |x_i \times P_{i,\mathcal{B}}|_{m_i}$
2: *attente pipeline 1 et écriture de* $\xi_{i,\mathcal{B}\to\mathcal{B}'}$ *dans* reg_S
3: **pour** j de 1 à n **faire**
4: $\quad v(j) = |v(j-1) + \xi_{i,\mathcal{B}\to\mathcal{B}'} \times M_{i,j,\mathcal{B}\to\mathcal{B}'} + C_{i,\mathcal{B}\to\mathcal{B}'}.SC|_{m'_i}$
5: **fin pour**
6: $|xi_{r,2\to1} = v(n) + x'_i \times P_{i,\mathcal{B}'}|_{m'_i}$
7: *attente pipeline 2*
8: $s'_i = |xi_{i,\mathcal{B}'\to\mathcal{B}} \times Q_{i,\mathcal{B}'}|_{m'_i}$ et écriture de $\xi_{i,\mathcal{B}'\to\mathcal{B}}$ dans reg_S
9: **pour** j de 1 à n **faire**
10: $\quad u(j) = |u(j-1) + \xi_{j,\mathcal{B}'\to\mathcal{B}} \times M_{i,j,\mathcal{B}'\to\mathcal{B}} + C_{i,\mathcal{B}'\to\mathcal{B}}.SC|_{m'_i}$.
11: **fin pour**
12: *attente pipeline 3*
13: **retourne** s'_i et $s_i = u(n)$

du bit actif(ou des w derniers bits pour le sliding windows). Enfin, suivant la complexité des algorithmes ou la régularité des opérations, un flot de contrôle classique (à base d'appel à des fonctions paramétrées et de boucles) peut être intéressant, pour limiter la taille du microcode, et épargner des mémoires. Encore une fois, la taille des boucles est fixe, et de même les paramètres des fonctions. Un adressage absolu des mémoires des Rower et relatif par rapport aux paramètres des fonctions est aussi nécessaire. La profondeur des piles nécessaires à l'exécution des fonctions de présentés dans cette thèse (4 suffit), une implémentation en registre est parfaitement acceptable. La figure 2.4 décrit ainsi la structure du séquenceur obtenu.

Naturellement, il ne s'agit pas de la seule possibilité de pilotage du séquenceur, mais celle-ci est extrêmement économe, puisqu'elle permet un pilotage de toutes les opérations du Cox-Rower pour un budget variant de 200 à 300 ALM en fonction des profondeurs des différentes piles.

2.5 Transformations binaire → RNS et RNS → binaire

Dans cette section, nous traitons du fonctionnement des entrée/sortie du cœur. Naturellement, il est possible de fournir les valeurs des bases RNS à l'extérieur du coprocesseur, soit pour traitement, soit éventuellement pour communication en l'état. Néanmoins, le format RNS n'est pas standard, et il est nécessaire d'être en mesure de réaliser efficacement les transformations de et vers la représentation binaire.

La transformation RNS → binaire présentée ici est une contribution de cette thèse, publiée dans [61].

TABLE 2.1 – Liste des précalculs nécessaires au changement de base, aux transformations binaire ↔ RNS

Algorithme	Nom	nombre	valeur	mémoire
Réduction	$P_{i,\mathcal{B}}$	n	$\|-\frac{m_i}{p\mathcal{M}(\mathcal{B})}\|_{m_i}$	principale
	$M_{i,j,\mathcal{B}\to\mathcal{B}'}$	n^2	$\|\frac{pm'_j}{m_i\mathcal{M}(\mathcal{B}')}\|_{m'_j}$	principale
	$C_{i,\mathcal{B}\to\mathcal{B}'}$	n	$\|-\frac{pm'_i}{\mathcal{M}(\mathcal{B}')}\|_{m'_i}$	ROM_2
	$P_{i,\mathcal{B}'}$	n	$\|\frac{m'_i}{\mathcal{M}(\mathcal{B})\mathcal{M}(\mathcal{B}')}\|_{m'_i}$	principale
	$Q_{i,\mathcal{B}'}$	n	$\|\frac{\mathcal{M}(\mathcal{B}')}{m'_i}\|_{m'_i}$	principale
	$M_{i,j,\mathcal{B}'\to\mathcal{B}}$	n^2	$\|\frac{\mathcal{M}(\mathcal{B}')}{m'_j}\|_{m_i}$	principale
	$C_{i,\mathcal{B}'\to\mathcal{B}}$	n	$\|-\mathcal{M}(\mathcal{B}')\|_{m_i}$	ROM_2
binaire → RNS	T_i	n	$\|\|\mathcal{M}(\mathcal{B})^2\|_p \times P_{i,\mathcal{B}}\|_{m_i}$	principale
	$T'i$	n	$\|\|\mathcal{M}(\mathcal{B})^2\|_p\|_{m'_i}$	principale
	R_i	n	$\|2^r\|_{m_i}$	principale
	R'_i	n	$\|2^r\|_{m'_i}$	principale
RNS → binaire	U'_i	n	$\|2^{-r} \times P_{i,\mathcal{B}'}\|_{m'_i}$	principale
	V'_i	n	$\|-2^{-r} \times P_{i,\mathcal{B}'}\|_{m'_i}$	principale

Algorithme 15 : Algorithme de transformation binaire vers RNS

ENTRÉE(S): $X = \mathbf{x}_1 + \mathbf{x}_2 2^r + \cdots \mathbf{x}_n 2^{r(n-1)}$
SORTIE(S): $S = |X\mathcal{M}(\mathcal{B})|_p$ dans $\mathcal{B}$ et $\mathcal{B}'$
1: $u_i(n) = \mathbf{x}_n \times T_{i,\mathcal{B}}$
2: $u'_i(n) = \mathbf{x}_n \times T'_{i,\mathcal{B}'}$
3: **pour** j de $n-1$ à 0 **faire**
4: $\quad v_i = |\mathbf{x}_j \times T_{i,\mathcal{B}}|_{m_i}$
5: $\quad u_i(j) = |u_i(j+1) \times R_i + v_i|_{m_i}$
6: $\quad v'_i = |fx_j \times T_{i,\mathcal{B}'}|_{m'_i}$
7: $\quad u'_i(j) = |u'_i(j+1) \times R'_i + v_i|_{m'_i}$
8: **fin pour**
9: sauter à la ligne 3 de l'algorithme 14.

2.5.1 Transformation binaire → RNS

Dans ce sens la transformation est triviale. En effet, supposant $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ la représentation en base 2^r de X, nous savons que :

$$|X|_{m_i} = |(\cdots(x_n 2^r + x_{n-1})2^r + \cdots)2^r + x_0)|_{m_i} \tag{2.7}$$

Par conséquent, l'algorithme 15 accompagné des précalculs de la table 2.1 donne le représentant de Montgomery de X dans la base $\mathcal{B}$ et $\mathcal{B}'$ en $6n\eta$ cycles de Cox-Rower (incluant les $\eta(2n+2)$ de la réduction de Montgomery RNS et $\eta(4n-2)$ de la boucle de l'algorithme).

2.5.2 Transformation RNS → binaire

Si la transformation binaire vers RNS est assez triviale à réaliser avec un Cox-Rower, la transformation inverse est plus complexe : dans [110], les auteurs proposent l'utilisation d'un module de transformation indépendant, tandis que Kawamura et al. [71], proposent une méthode rapide utilisant la propagation de retenue d'un Rower à l'autre, soit du

Algorithme 16 : Algorithme de transformation RNS vers binaire

Entrée(s): $X = (x_1, \cdots x_n)$ et $(x'_1, \cdots, x'_n)$ dans $\mathcal{B}$ et $\mathcal{B}'$
Sortie(s): $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ représentation en base 2^r de X.

1: Exécuter l'algorithme 14
2: $u_i = x_i$ et $u'_i = x_i$
3: **pour** j de 1 à n **faire**
4: $\quad \mathbf{x}_j = u_0$
5: $\quad u'_i = |u'_i \times V'_i|_{m'_i}$
6: $\quad u'_i = |u'_i + u_0 \times U'_i|_{m'_i}$
7: $\quad$ sauter à la ligne 8 de l'algorithme 14
8: **fin pour**

matériel supplémentaire par rapport à celui qui a été défini dans ce chapitre, matériel uniquement nécessaire pendant la transformation finale à chaque calcul.

Nous verrons dans les chapitres suivants que la transformation RNS vers binaire a une complexité très largement inférieure à celle des traitements imposés par la cryptographie. Par conséquent, il est possible d'utiliser un algorithme sous optimal en terme de complexité si le temps de traitement reste faible par rapport au reste des traitements. C'est ce que je propose ici.

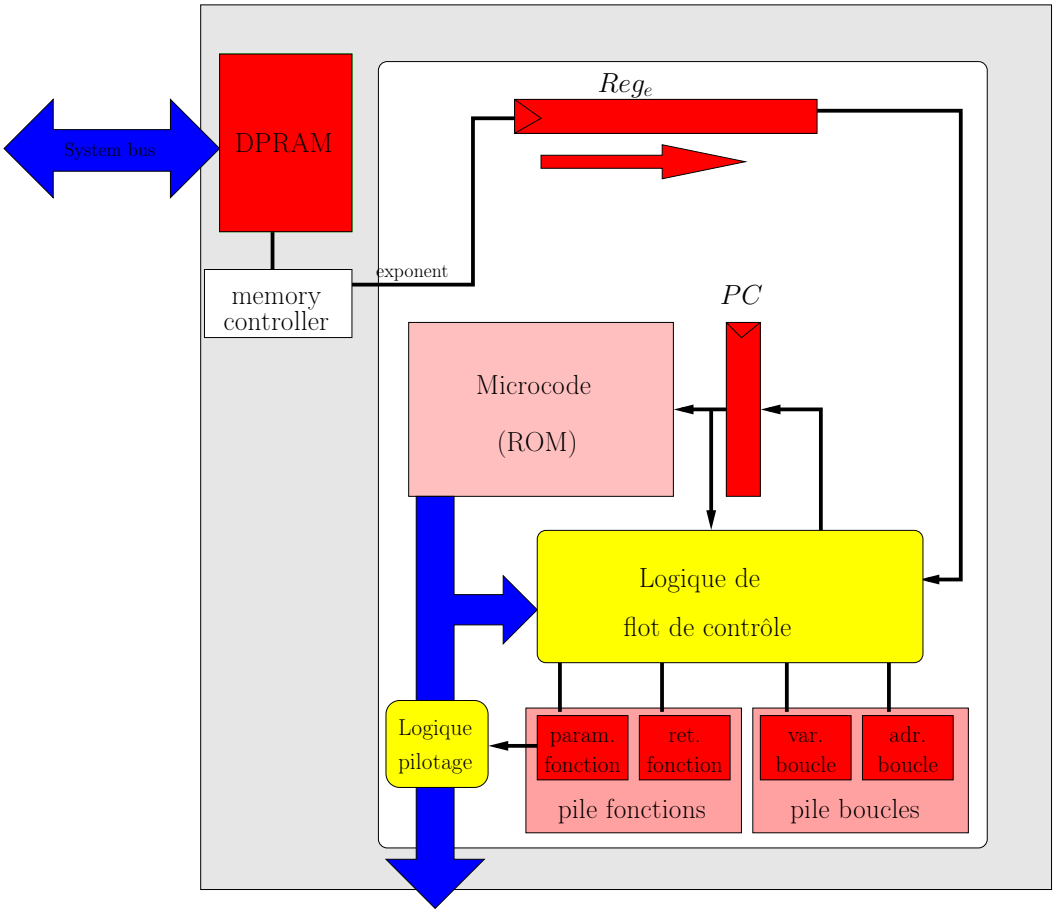
L'algorithme 16 peut être entièrement exécuté sur un Cox-Rower sans ajout de matériel. L'idée sous jacente de cet algorithme est que le mot de poids faible de la représentation binaire est égal à la valeur du canal $m_1 = 2^r$. Ainsi nous considérons les bases RNS suivantes : $\mathcal{B}'$ défini comme précédemment et $\check{B} = (2^r)$. A l'issue de la transformation de Montgomery inverse, la valeur d'entrée de l'algorithme 16 est inférieure à $3p$. Ainsi nous réalisons le traitement suivant :

– par changement de base entre $\check{\mathcal{B}}$ et $\mathcal{B}'$ (qui est trivial) nous obtenons la valeur $(|X|_{2^r})_{\mathcal{B}'}$ puis $\left(\frac{X-|X|_{2^r}}{2^r}\right)_{\mathcal{B}'} = \left(\lfloor \frac{X}{2^r} \rfloor\right)_{\mathcal{B}'}$.

– par changement de base entre $\mathcal{B}'$ et $\check{\mathcal{B}}$ nous obtenons le mot de poids faible suivant.

Il est aisé de noter que le changement de base entre $\check{\mathcal{B}}$ et $\mathcal{B}'$ respecte les conditions décrites dans la section 2.3.4 sur les valeurs respectives des modules des bases. Par conséquent le changement de base de Kawamura peut être exécuté sans erreur d'approximation.

FIGURE 2.4 – Module de pilotage du Cox-Rower : un séquenceur spécifiquement dessiné pour les opérations cryptographiques



Chapitre 3

Implémentation efficace de la multiplication scalaire sur courbe elliptique en RNS à tous les niveaux de sécurité

Sommaire

3.1 Introduction 45
3.2 Arithmétique des courbes elliptiques, impacts du RNS 46
3.2.1 Courbes elliptiques : définition 46
3.2.2 Diviseurs et structure de groupe 47
3.2.3 Formules d'addition et doublement sur une courbe elliptique définie sur $\mathbb{F}_p$ 48
3.2.4 Impact du RNS sur l'arithmétique des courbes 50
3.2.5 Choix de base 51
3.3 Parallélisation et pipeline 52
3.3.1 Optimisation des variables temporaires, dépendance de données et parallélisation 52
3.3.2 Structure du pipeline 54
3.3.3 Sécurité 57
3.3.4 Conclusion 58
3.4 Performances et comparaisons 58
3.4.1 Résultat d'implémentation pour l'ensemble des niveaux de sécurité 58
3.4.2 Comparaisons 59
3.5 Conclusion 61

3.1 Introduction

Dans ce chapitre, je démontre l'intérêt du RNS et de l'architecture Cox-Rower dans le cas de la multiplication scalaire sur courbes elliptiques. Le grand intérêt de la représentation RNS est son arithmétique qui permet de travailler parallèlement sur des grands nombres (voir chapitre 2). On peut alors supposer que l'utilisation du RNS dans le cadre des courbes elliptiques est moins intéressante que dans le cas du RSA, en raison de la taille de la caractéristique du corps sous-jacent. Je démontre dans ce chapitre que cet inconvénient

est largement compensé par le fait que le RNS permet de réduire de manière importante la complexité des opérations d'addition et doublement de points dans le cas général (p n'étant pas un nombre de pseudo-Mersenne) et que par ailleurs, le parallélisme présenté par ces formules permet une utilisation à presque 100% du pipeline dans le cas d'une implémentation même à des très petites tailles de courbe (160 bits). J'ai implémenté ce circuit sur des FPGA de la gamme Stratix et Stratix 2, avec des résultats très prometteurs, et ce à tous les niveaux de sécurité (de 160 à 512 bits).

3.2 Arithmétique des courbes elliptiques, impacts du RNS

Dans cette section, je traite de l'impact du choix de l'utilisation du RNS dans l'arithmétique des courbes elliptiques.

3.2.1 Courbes elliptiques : définition

Cette section est une synthèse de [115] introduisant les concepts nécessaires à cette thèse. Je ne m'intéresse qu'aux corps finis même si les résultats ci-dessous sont corrects sur tous les corps parfaits.

Soit $\mathbb{F}_q$ un corps fini avec $q = p^k$ et $\overline{\mathbb{F}_q}$ sa clotûre algébrique.

Je définis la relation d'équivalence $\sim$ entre les éléments de $\overline{\mathbb{F}_q}^3$ telle que

$$(x_1, y_1, z_1) \sim (x_2, y_2, z_2) \Leftrightarrow \exists \lambda \in \overline{\mathbb{F}_q}\ x_1 = \lambda x_2, y_1 = \lambda y_2, z_1 = \lambda z_2$$

Soit $\mathbb{P}^2(\overline{\mathbb{F}_q})$ l'espace quotient de $\overline{\mathbb{F}_q}^3$ par $\sim$ (et $\mathbb{P}^2(\mathbb{F}_q)$ sa restriction sur $\mathbb{F}_q$). La notion d'espace projectif permet de définir un élément essentiel pour l'étude et les opérations sur les courbes elliptiques qui est le point à l'infini.

Définition 5 *Soit $\mathbb{F}_q[X, Y, Z]$ l'espace polynomial trivarié de $\mathbb{F}_q$ et soit $f_\mathcal{C} \in \mathbb{F}_q[X, Y, Z]$. $f_\mathcal{C}$ est dit homogène de degré d si*

$$\forall \lambda \in \overline{\mathbb{F}_q}\ f(\lambda X, \lambda Y, \lambda Z) = \lambda^d f(X, Y, Z) \tag{3.1}$$

Définition 6 *Soit $f \in \mathbb{F}_q[X, Y, Z]$ premier, homogène et de degré 3 tel que $f(0, 1, 0) = 0$. Une courbe elliptique $\mathcal{C}$ peut être vue comme l'ensemble des points $\mathbf{P} = (x, y, z) \in \mathbb{P}^2(\mathbb{F}_q)$ tel que $f(x, y, z) = 0$. Le polynôme f est noté $f_\mathcal{C}$.*

Ainsi, une courbe elliptique est définie par l'équation de Weierstraβ suivante :

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{3.2}$$

[115, chapitre 3 section 3 proposition 3.1] démontre que toute courbe algébrique de genre 1 non vide est isomorphe (dans le sens d'isomorphisme entre variétés algébriques) à une courbe sous la forme de l'équation 3.2, dite équation de Weierstraβ.

Théorème 3 *Si $p \notin \{2, 3\}$ alors toute courbe elliptique $\mathcal{C}$ est isomorphe à une courbe $\mathcal{C}'$ définie par le polynôme suivant :*

$$f'(X, Y, Z) = Y^2 - X^3 - a_4XZ^2 - a_6Z^3 \tag{3.3}$$

Cette équation est dite équation de Weierstraβ réduite.

On considère la fonction $\mathcal{P}$ de tous les éléments de $\mathbb{P}^2(\mathbb{F}_q)$ où $Z \neq 0$ vers ${\mathbb{F}_q}^2$ qui à (X, Y, Z) associe $(x, y) = (X/Z, Y/Z)$. Nous pouvons constater que l'image de la courbe elliptique $\mathcal{C}$ définie par l'équation 3.2 par $\mathcal{P}$ est la courbe d'équation :

$$y^2 + (a_1 x + a_3)y = x^3 + a_2 x^2 + a_4 x + a_6 \tag{3.4}$$

Ainsi une courbe elliptique est une courbe dans le plan constitué par l'ensemble des points respectant l'équation 3.4, auquel on ajoute un point, dit point à l'infini ou $\mathcal{O}$, de coordonnées projectives $(0; 1; 0)$. Pour tout point $\mathbf{P} \neq \mathcal{O}$, les coordonnées (x, y) sont les coordonnées affines de $\mathbf{P}$ tandis que (X, Y, Z) sont les coordonnées projectives. De même, si $p \notin \{2, 3\}$ alors l'équation affine devient

$$y^2 = x^3 + a_4 x + a_6 \tag{3.5}$$

Par la suite je confondrai $\mathcal{C}$ avec $\mathcal{P}(\mathcal{C})$.

Définition 7 *Soit $\mathcal{C}$ une courbe elliptique et $\mathbf{P} \neq \mathcal{O}$ un point de $\mathcal{C}$. On dit que $\mathcal{C}$ est singulière en $\mathbf{P} = (x; y)$ si*

$$\frac{\partial f_{\mathcal{C}}}{\partial x} = \frac{\partial f_{\mathcal{C}}}{\partial y} = 0 \tag{3.6}$$

Si $\mathcal{C}$ ne possède aucun point singulier, on dit que $\mathcal{C}$ est lisse.

Si $p \notin \{2; 3\}$, une courbe elliptique est lisse si et seulement si $27a_4^3 + 4a_6^2 \neq 0$.

3.2.2 Diviseurs et structure de groupe

Définition 8 *Soit $\mathcal{C}$ une courbe elliptique lisse. Un diviseur D de $\mathcal{C}$ est une somme finie formelle de points $(\mathbf{P}_1; \cdots; \mathbf{P}_n)$ de $\mathcal{C}$. Soit $(\boldsymbol{\alpha}_1; \cdots; \boldsymbol{\alpha}_n) \in \mathbb{Z}^*$.*

$$D = \boldsymbol{\alpha}_1(\mathbf{P}_1) + \cdots \boldsymbol{\alpha}_n(\mathbf{P}_n) \tag{3.7}$$

L'ensemble $Div(\mathcal{C})$ des diviseurs de $\mathcal{C}$ est un groupe abélien. Par ailleurs, l'ensemble fini des diviseurs tels que $\sum_{i=0}^{n} \boldsymbol{\alpha}_i = 0$ est un sous groupe abélien de $Div(\mathcal{C})$ noté $Div^0(\mathcal{C})$.

Théorème 4 *Soit $F(X, Y, Z)$ l'ensemble des fonctions rationnelles $f(X, Y, Z)/g(X, Y, Z)$ sur $\overline{\mathbb{F}_q}$ telles que f et g soient homogènes et de même degré. Soit $\mathcal{C}$ une courbe elliptique, $f_{\mathcal{C}}$ son équation de Weierstraβ, et $I(f)$ l'idéal de $\overline{\mathbb{F}_q}[X, Y, Z]$ des polynômes homogènes engendré par $f_{\mathcal{C}}$. L'ensemble quotient des éléments de $F(X, Y, Z)$ tel que $g(X, Y, Z) \notin I(f)$ par la relation d'équivalence $f/g \equiv f'/g' \Leftrightarrow f'g - fg' \in I(f_{\mathcal{C}})$ est un corps, appelé corps de fonction de la courbe $\mathcal{C}$ et noté $\overline{\mathbb{F}_q}(\mathcal{C})$. L'ensemble des fonctions rationnelles de $\overline{\mathbb{F}_q}(\mathcal{C})$ telles que dans leur classe d'équivalence il existe une fonction rationnelle avec tous ses termes dans $\mathbb{F}_q$ forme un sous corps de $\overline{\mathbb{F}_q}(\mathcal{C})$ noté $\mathbb{F}_q(\mathcal{C})$.*

Une démonstration dans un cadre plus général des variétés algébriques se trouve dans [115, chapitre 1].

Théorème 5 *Pour tout élément f de $\overline{\mathbb{F}_q}(\mathcal{C})$ non nul, on peut associer un diviseur $D = \boldsymbol{\alpha}_1(\mathbf{P}_1) + \cdots \boldsymbol{\alpha}_n(\mathbf{P}_n) \in Div^0(\mathcal{C})$ tel que si $\boldsymbol{\alpha}_i > 0$ alors $\mathbf{P}_i$ est un zéro de f d'ordre $\boldsymbol{\alpha}_i$ et si $\boldsymbol{\alpha}_i < 0$, $\mathbf{P}_i$ est un pôle de f d'ordre a_i. L'ensemble des diviseurs associés à une fonction f de $\overline{\mathbb{F}_q}(\mathcal{C})$ forme un sous groupe abélien de $Div^0(\mathcal{C})$ isomorphe à $\overline{\mathbb{F}_q}(\mathcal{C})$ muni de la multiplication. Ce sous groupe est appelé ensemble des diviseurs principaux de $\mathcal{C}$ ou $\texttt{Princ}_{\mathcal{C}}$.*

Le lecteur pourra se référer à [115, chapitre 2 section 1.1] pour la définition formelle de l'ordre d'un zéro ou d'un pôle de f en $\mathbf{P}$.

Définition 9 *Le sous groupe quotient de $Div^0(\mathcal{C})$ par $\texttt{Princ}(\mathcal{C})$ est appelé jacobienne de $\mathcal{C}$ et noté $Pic^0(\mathcal{C})$.*

C'est en réalité sur ce groupe que l'on réalise les opérations cryptographiques.

Théorème 6 *Pour tout diviseur $D \in Div^0(\mathcal{C})$, $\exists \mathbf{P} \in \mathcal{C}$ tel que $D = (\mathbf{P}) - (\mathcal{O}) + D'$ avec $D' \in Pic^0(\mathcal{C})$.*

Ce théorème fondamental dit en réalité que la jacobienne de la courbe est isomorphe à la courbe, et que l'on peut donc définir une loi de groupe sur les points de la courbe elliptique qui correspond à la loi de groupe des diviseurs associés, et dont le point à l'infini est l'élément neutre. Voir [115] pour une démonstration de ce théorème.

3.2.3 Formules d'addition et doublement sur une courbe elliptique définie sur $\mathbb{F}_p$

Je me plaçe maintenant dans le cas ou $p = q \notin \{2, 3\}$.

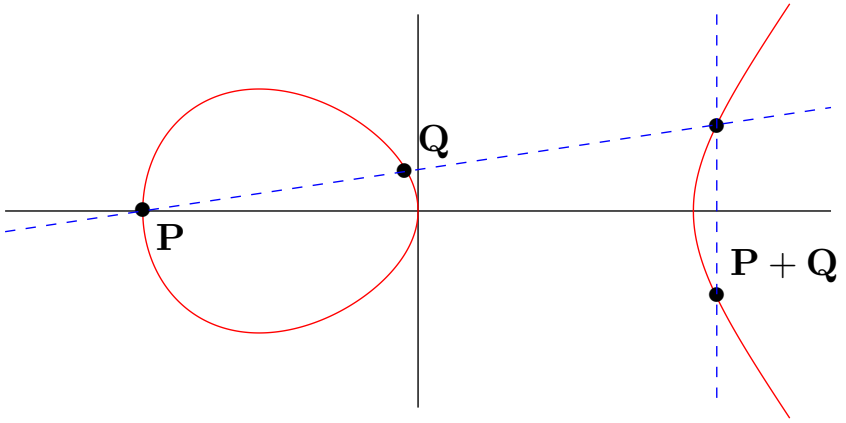
L'addition de $(\mathbf{P})$ et $(\mathbf{Q})$ peut se réaliser en utilisant les droites d'équation $g(X) = aX + bY + cZ$. En effet, soit $D = (\mathbf{Q}) + (\mathbf{P}) - 2(\mathcal{O})$. Par la nature de la définition de $f_{\mathcal{C}}$, la droite passant par $\mathbf{P}$ et $\mathbf{Q}$ coupe systématiquement la courbe en un unique troisième point $\mathbf{R}$. Le diviseur $D' = (\mathbf{Q}) + (\mathbf{P}) + (\mathbf{R}) - 3(\mathcal{O})$ est donc le diviseur principal de la droite, et par conséquent $D = -(\mathbf{R}) + (\mathcal{O}) + D'$. Or, le diviseur de la droite verticale $d(X, Y, Z) = X - x_{\mathbf{R}}Z$ possède elle même $\mathbf{R}$ et le point $\mathcal{O}$, son diviseur associé est donc $D'' = (\mathbf{R}) + (-\mathbf{R}) - 2\mathcal{O}$ (le point $-\mathbf{R}$ étant défini par $X_{-\mathbf{R}} = X_{\mathbf{R}}$ et $Y_{-\mathbf{R}} = -Y_{\mathbf{R}}$). Donc $D = (-\mathbf{R}) - (\mathcal{O}) + D' + D''$. Dans la jacobienne, on a donc $(\mathbf{Q}) + (\mathbf{P}) - 2(\mathcal{O}) = (-\mathbf{R}) - (\mathcal{O})$ car $D' + D'' \in \texttt{Princ}(\mathcal{C})$ En remplaçant $(\mathbf{Q}) + (\mathbf{P}) - 2(\mathcal{O})$ par $(-\mathbf{R}) - (\mathcal{O})$, nous venons de réaliser la construction classique de la corde et la verticale pour additionner les points sur une courbe elliptique. On peut noter par ailleurs que les points de la courbe définis dans $\mathbb{P}^2(\mathbb{F}_q)$ forment un sous groupe de $Pic^0(\mathcal{C})$, car les droites ayant 2 points d'intersection avec la courbe dans $\mathbb{P}^2(\mathbb{F}_q)$ ont leur troisième intersection aussi dans $\mathbb{P}^2(\mathbb{F}_q)$.

La figure 3.1 donne une représentation géométrique de l'addition sur une courbe elliptique dans le cas des réels. La définition analytique de l'addition est donnée ci-dessous $\mathbf{P} = (X_{\mathbf{P}}; Y_{\mathbf{P}}; Z_{\mathbf{P}})$ et $\mathbf{Q} = (X_{\mathbf{Q}}; Y_{\mathbf{Q}}; Z_{\mathbf{Q}})$ sur $\mathcal{C}_{p,a_4,a_6}$ alors les point $\mathbf{P} + \mathbf{Q}$ et $2\mathbf{P}$ ont les coordonnées suivantes

$$
\begin{array}{rcl}
X_{\mathbf{P}+\mathbf{Q}} & = & (X_{\mathbf{Q}}Z_{\mathbf{P}} - X_{\mathbf{P}}Z_{\mathbf{Q}})\cdot \\
 & & \big((Y_{\mathbf{Q}}Z_{\mathbf{P}} - Y_{\mathbf{P}}Z_{\mathbf{Q}})Z_{\mathbf{P}}Z_{\mathbf{Q}} - (X_{\mathbf{Q}}Z_{\mathbf{P}} - X_{\mathbf{P}}Z_{\mathbf{Q}})^2(X_{\mathbf{Q}}Z_{\mathbf{P}} + Z_{\mathbf{P}}X_{\mathbf{Q}})\big) \\
Y_{\mathbf{P}+\mathbf{Q}} & = & (Y_{\mathbf{P}}Z_{\mathbf{Q}} - Y_{\mathbf{Q}}Z_{\mathbf{P}}) + \\
 & & (X_{\mathbf{Q}}Z_{\mathbf{P}} - X_{\mathbf{P}}Z_{\mathbf{Q}})\left((X_{\mathbf{Q}}Z_{\mathbf{P}} - X_{\mathbf{P}}Z_{\mathbf{Q}})(2X_{\mathbf{P}}Z_{\mathbf{Q}} + X_pQZ_{\mathbf{Q}}) - Y_{\mathbf{P}}Z_{\mathbf{Q}}\right) \\
Z_{\mathbf{P}+\mathbf{Q}} & = & (X_{\mathbf{Q}}Z_{\mathbf{P}} - X_{\mathbf{P}}Z_{\mathbf{Q}})^3 Z_{\mathbf{P}}Z_{\mathbf{Q}} \\
X_{2P} & = & 2Y_{\mathbf{P}}Z_{\mathbf{P}}\left((a_4Z_{\mathbf{P}}^2 + 3X_{\mathbf{P}}^2)^2 - 8Y_{\mathbf{P}}^3Z_{\mathbf{P}}\right) \\
Y_{2P} & = & (a_4Z_{\mathbf{P}}^2 + 3X_{\mathbf{P}}^2)\left(2X_{\mathbf{P}}Y_{\mathbf{P}}^2Z_{\mathbf{P}} - (a_4Z_{\mathbf{P}}^2 + 3X_{\mathbf{P}}^2)^2\right) - 8Y_{\mathbf{P}}^4Z_{\mathbf{P}}^2 \\
Z_{2P} & = & (2Y_{\mathbf{P}}Z_{\mathbf{P}})^3
\end{array}
$$

Les travaux de Brier et Joye dans [32] et parallèlement de Izu et Takagi [73] ont mis en évidence une formule ne mettant pas en jeu les valeurs de $Y_{\mathbf{P}}$ et $Y_{\mathbf{Q}}$ mais la coordonnée affine $x_{\mathbf{P}-\mathbf{Q}}$ pour calculer $X_{\mathbf{P}+\mathbf{Q}}$ et $Z_{\mathbf{P}+\mathbf{Q}}$. Ces formules sont générales, et ne nécessitent

FIGURE 3.1 – Addition sur une courbe elliptique définie sur les réels



pas de formes courbes particulières. En effet la généricité reste notre objectif prioritaire dans ce travail.

Les formules sont données dans le tableau 3.1 : Cette formule a l'avantage de ne pas

TABLE 3.1 – Formules d'addition et doublement sans l'ordonnée du point

$$\begin{array}{rcl} X_{\mathbf{P}+\mathbf{Q}} & = & (Z_{\mathbf{P}}X_{\mathbf{Q}} + Z_{\mathbf{Q}}X_{\mathbf{P}})\,(X_{\mathbf{P}}X_{\mathbf{Q}} + 2a_4 Z_{\mathbf{P}}Z_{\mathbf{Q}} + 2x_{\mathbf{P}-\mathbf{Q}}(Z_{\mathbf{P}}X_{\mathbf{Q}} - X_{\mathbf{P}}Z_{\mathbf{Q}})) \\ & & +2a_6 Z_{\mathbf{P}}^2 Z_{\mathbf{Q}}^2 \\ Z_{\mathbf{P}+\mathbf{Q}} & = & (X_P Z_{\mathbf{Q}} - X_Q Z_{\mathbf{P}})^2 \\ X_{2P} & = & -8a_6 Z_{\mathbf{P}}^3 X_{\mathbf{P}} + (X_{\mathbf{P}}^2 - a_4 Z_{\mathbf{P}}^2)^2 \\ Z_{2P} & = & 4Z_{\mathbf{P}}(X_{\mathbf{P}}^3 + a_4 X_{\mathbf{P}} Z_{\mathbf{P}}^2 + a_6 Z_{\mathbf{P}}^3) \end{array}$$

nécessiter la coordonnée Y, et donc de ne pas avoir besoin de la calculer. En revanche la coordonnée $x_{\mathbf{P}-\mathbf{Q}}$ doit être connue, il est donc nécessaire de choisir correctement l'algorithme de multiplication scalaire. L'échelle de Montgomery [78] (algorithme 17) permet de résoudre ce problème. En effet, sur l'algorithme 17, il est aisé de voir que les variables locales $\mathbf{R}$ et $\mathbf{S}$ sont définies de telle manière que $\mathbf{S} = \mathbf{R} + \mathbf{G}$. Par conséquent, la calcul de $\mathbf{R} + \mathbf{S}$ nécessite $x_{\mathbf{R}-\mathbf{S}} = x_{\mathbf{G}}$. Ainsi la quantité supplémentaire de calculs due à un doublement et une addition par bit de k est compensée par le fait que $Y_{\mathbf{R}}$ et $Y_{\mathbf{S}}$ ne sont pas calculés. Ainsi, l'échelle de Montgomery reste relativement compétitive par rapport à d'autres algorithmes de multiplication scalaire, comme le sliding Windows ou le recodage Non Adjacent Form (NAF) du scalaire k.

En revanche, il peut s'avérer utile de retrouver la valeur de $y_{k\mathbf{G}}$ à l'issue du calcul. Cette valeur peut être retouvée grâce à l'équation suivante :

$$y_{k\mathbf{G}} = (2a_6 + (a_4 + x_{\mathbf{G}}x_{k\mathbf{G}})(x_{\mathbf{G}} + x_{k\mathbf{G}}) - x_{[k+1]\mathbf{G}}(x_{\mathbf{G}} - x_{k\mathbf{G}})^2)(2y_{\mathbf{G}})^{-1} \tag{3.8}$$

En plus de l'avantage calculatoire que donne l'utilisation de cet algorithme, celui-ci est naturellement résistant à la SPA.

Algorithme 17 : $Montg - ladder(k, \mathbf{G}, \mathbf{C}_{p,a_4,a_6})$

Entrée(s): $k = \sum k_i.2^i, \mathbf{G}$ a point of $\mathbf{C}_{p,a_4,a_6}$
Sortie(s): $\mathbf{R} = [k]\mathbf{G}$
1: $\mathbf{R} \leftarrow \mathcal{O}$; $\mathbf{S} \leftarrow \mathbf{G}$
2: **pour** i de $\log_2(k)$ à 0 **faire**
3: **si** $(k_i = 0)$ **alors**
4: $\mathbf{R} \leftarrow 2R$; $\mathbf{S} \leftarrow \mathbf{R} + \mathbf{S}$
5: **sinon**
6: $\mathbf{S} \leftarrow 2S$; $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{S}$
7: **fin si**
8: **fin pour**
9: **retourne** $\mathbf{R}$

3.2.4 Impact du RNS sur l'arithmétique des courbes

Comme je l'ai indiqué dans le chapitre 2, le RNS n'apporte aucun avantage dans le cas d'une exponentiation en terme de complexité.

En considérant un entier de n mots, et en ne considérant que les multiplications de mots machines, une multiplication en représentation binaire coûte n^2, tandis que la réduction coûte un prix équivalent à $n^2 + n$. Les travaux du groupe GMP [3] ont démontré que pour des tailles cryptographiques usuelles des courbes elliptiques, les techniques d'interpolation comme Karatsuba ou Toom-Cook ne sont pas utiles, et l'algorithme de multiplication utilisé est le classique "schoolbook".

En RNS la multiplication coûte n multiplications modulaire par les entiers de la base, tandis que la réduction coûte $2n^2+3n$. Par conséquent dans le cas d'une exponentiation modulaire, du fait que chaque multiplication ou carré est suivi d'une réduction, la complexité est globalement identique.

En revanche, cela n'est plus vrai dans le cas des courbes elliptiques. En effet, l'apparition d'additions dans les formules d'arithmétique des courbes permet de profiter pleinement du mécanisme de la réduction fainéante ou *lazy reduction*. En effet, tout motif de type $X_0X_1 + X_2X_3 + \cdots + X_{l-1}X_l$ ne nécessite qu'une seule réduction modulaire. Si cela est vrai aussi en représentation binaire, le faible coût de la multiplication en RNS rend cette possibilité intressante. Une telle opération coûte $n^2 + (2l + 3)n$ multiplications en RNS, au lieu de $(l + 1)n^2 + n$ en binaire.

Le tableau 3.2 réarrange ainsi les formules 3.1 sous la forme permettant de limiter au maximum le nombre de réductions nécessaires. Elles sont directement issues de [13], contribution qui étudie l'ensemble des formules apportant une résistance à la SPA pour les courbes de genre 1 (incluant les courbes ayant une représentation permettant une arithmétique plus rapide). De manière assez étonnante, les formules de doublement nécessitent une réduction de plus que les formules d'addition (7 au lieu de 6).

Enfin, les formules de l'algorithme 18 donnent les formules pour obtenir $(x_{k\mathbf{G}}, y_{k\mathbf{G}})$ à partir des valeurs de $x_{\mathbf{G}}, y_{\mathbf{G}}, X_{k\mathbf{G}}, X_{(k+1)\mathbf{G}}, Z_{k\mathbf{G}}, Z_{(k+1)\mathbf{G}}$. Ces formules réduisent le nombre d'inversions à 1 via l'astuce de Montgomery $A^{-1} = (AB)^{-1}B$ et $B^{-1} = (AB)^{-1}A$.

TABLE 3.2 – Formules des lois de groupe limitant le nombre de réductions

$\mathbf{P}+\mathbf{Q}$	$2\mathbf{P}$
$A \leftarrow Z_P X_{\mathbf{Q}} + X_P Z_{\mathbf{Q}}$	$E \leftarrow Z_p^2$
$B \leftarrow 2X_P X_{\mathbf{Q}}$	$F \leftarrow 2X_P Z_{\mathbf{P}}$
$C \leftarrow 2Z_P Z_{\mathbf{Q}}$	$G \leftarrow X_{\mathbf{P}}^2$
$D \leftarrow a_4 A + a_6 C$	$H \leftarrow -4a_6 E$
$Z_{\mathbf{P}+\mathbf{Q}} \leftarrow A^2 - BC$	$I \leftarrow a_4 E$
$X_{\mathbf{P}+\mathbf{Q}} \leftarrow BA + CD + 2x_{\mathbf{P}-\mathbf{Q}} Z_{\mathbf{P}+\mathbf{Q}}$	$X_{2P} \leftarrow FH + (G-I)^2$
	$Z_{2P} \leftarrow 2F(G+I) - EH$

Algorithme 18 : Formules pour obtenir les coordonnées affines de $k\mathbf{G}$ à l'issue de l'algorithme d'exponentation

ENTRÉE(S): $x_{\mathbf{G}}, y_{\mathbf{G}}, X_{k\mathbf{G}}, X_{(k+1)\mathbf{G}}, Z_{k\mathbf{G}}, Z_{(k+1)\mathbf{G}}$
SORTIE(S): $x_{k\mathbf{G}}, y_{k\mathbf{G}}$
1: $A \leftarrow x_{\mathbf{G}} Z_{k\mathbf{G}}$
2: $B \leftarrow a_4 Z_{k\mathbf{G}} + x_{\mathbf{G}} X_{k\mathbf{G}}$
3: $C \leftarrow Z_{k\mathbf{G}} Z_{(k+1)\mathbf{G}}$
4: $D \leftarrow 2y_{\mathbf{G}}$
5: $E \leftarrow Z_{(k+1)\mathbf{G}} B$
6: $F \leftarrow 2a_6 C$
7: $G \leftarrow (A - X_{k\mathbf{G}})^2 C$
8: $X_{\mathbf{R}} \leftarrow D X_{k\mathbf{G}}$
9: $Z_{\mathbf{R}} \leftarrow D Z_{k\mathbf{G}}$
10: $Y_{\mathbf{R}} \leftarrow X_{(k+1)\mathbf{G}} G + E X_{k\mathbf{G}} + EA + FZ$
11: **retourne** $x_{\mathbf{R}} \leftarrow X_{\mathbf{R}} Z_{\mathbf{R}}^{-1}$ et $y_{\mathbf{R}} \leftarrow Y_{\mathbf{R}} Z_{\mathbf{R}}^{-1}$

3.2.5 Choix de base

Le choix de la base est naturellement essentiel. On considère deux bases RNS de taille n appelées $\mathcal{B}$ et $\mathcal{B}'$. Je propose ici de dimensionner n en fonction du choix de r la taille des bus du Cox-Rower (qui correspond à la taille des éléments des bases) et de la taille de la courbe. Pour cela, il est d'abord nécessaire de dimensionner la valeur α du théorème 2. Au regard des formules d'addition et de doublement données dans le tableau 3.2, on peut conclure que $\alpha = 45$ est suffisant. En effet, la valeur potentiellement maximale à réduire est la valeur $Z_{2P} = 2F(G+I) - EH$. Comme F,G, H et I sont inférieurs à $3p$, $2F(G+I)$ est au plus égal à $36p^2$. L'algorithme 11 ne pouvant traiter que des entrées positives, et comme il n'est pas possible de vérifier si $2F(G+I) > EH$ sans utiliser un algorithme de comparaison, il est nécessaire de calculer $(3p-E)H$ qui est positif et inférieur à $9p^2$. Par conséquent, $\alpha = 45$ est suffisant.

Dans le cas d'une implémentation de FPGA, je privilégie $r \leq 18$ ou $r \leq 36$. En effet, ce choix permet de bénéficier complètement des multiplieurs 18×18 embarqués dans les familles Xilinx ou Altera.

Dans le tableau 3.3, je donne les caractéristiques des bases en fonction du niveau de sécurité de la courbe et du choix de la taille des multiplieurs. r_ε est défini comme suit :

$$r_\varepsilon = \max_{m \in \mathcal{B} \cup \mathcal{B}'} \lceil log(2^r - \varepsilon) \rceil$$

Celui-ci doit nécessairement être inférieur à $r/2$, afin de limiter la complexité de la réduction modulaire (les éléments de la base sont des pseudo-Mersenne comme définis dans la section 1.2.1). Par conséquent, on peut constater que le choix $r = 18$ n'est pas interdit même au plus haut niveau de sécurité proposé (256 bits).

TABLE 3.3 – Dimensionnement des bases RNS en fonction du niveau de sécurité souhaité

sécurité	80	96	128	192	256	80	96	128	192	256
$log_2(p)$	160	192	256	384	512	160	192	256	384	512
r	17	18	18	18	18	34	33	33	36	35
n	10	11	15	22	29	5	6	8	12	15
r_ε	7	7	8	8	9	5	6	6	7	8

3.3 Parallélisation et pipeline

Cette section a pour but de partir de l'arithmétique définie dans la section précédente et de définir les éléments du Cox-Rower. Si la structure générale du Cox-Rower ne diffère pas par rapport à la présentation générale de la section 2.4, 2 éléments doivent être dimensionnés :

- la structure de la mémoire principale : celle-ci doit posséder de la mémoire en quantité suffisante pour la réalisation de la multiplication scalaire. Il n'est en revanche pas nécessaire que l'intégralité des valeurs contenues dans cette mémoire soit accessible en écriture et en lecture sur les 2 ports de cette mémoire.
- la structure du pipeline : la structure du Cox-Rower apporte une bonne garantie que les chemins critiques seront dans les unités de calcul. En effet, il n'existe dans le reste du circuit que peu de logique. Il convient donc de soigner le pipeline afin de garantir une fréquence de fonctionnement la plus rapide possible.

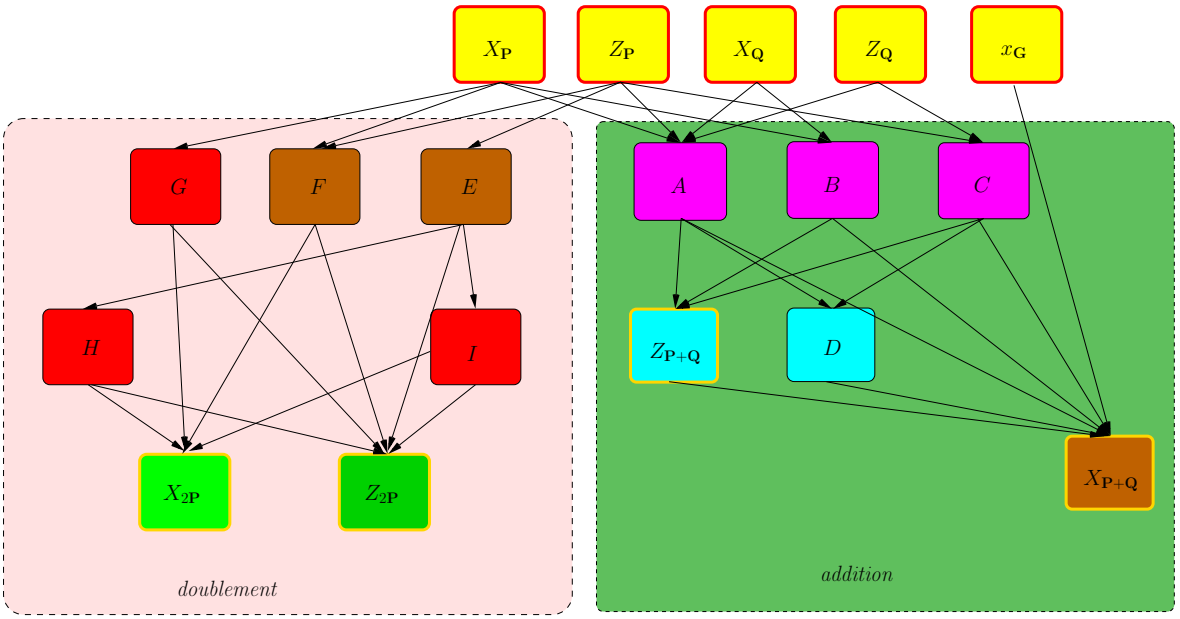
Les deux questions sont liées par le problème de la parallélisation. En effet, une profondeur du pipeline trop importante nécessite plus de parallélisme (i.e moins de dépendance) dans les équations pour obtenir un fort taux d'occupation. D'un autre côté, le parallélisme peut obliger à augmenter artificiellement le nombre de variables locales nécessaires. Je ne propose pas ici une étude complète, mais propose un compromis.

3.3.1 Optimisation des variables temporaires, dépendance de données et parallélisation

La contrainte prise est le traitement atomique de chaque étape de la boucle débutant à la ligne 2 de l'algorithme 17. Si cette contrainte peut être restrictive, elle limite substantiellement la complexité du séquenceur. La levée de cette contrainte reste à être étudiée. La figure 3.2 indique les dépendances entre les variables temporaires. Il est supposé qu'à chaque étape, le point **P** est doublé et additionné avec **Q**. **P** sera alors tantôt **R**, tantôt **S** de l'algorithme 17.

On peut déduire de l'étude de cette figure que seuls 8 registres sont nécessaires, et que 3 variables temporaires peuvent être calculées en même temps, excepté lors de l'étape 2 et 5. Les variables temporaires sont ainsi réparties dans 8 registres différents de la manière suivante :

FIGURE 3.2 – Dépendance des variables temporaires et parallélisme



étape	0	1	2	3	4	5
Calcul		A B C	D $Z_{\mathbf{P+Q}}$	$X_{\mathbf{P+Q}}$ E F	G H I	$X_{\mathbf{2P}}$ $Z_{\mathbf{2P}}$
reg 1	$X_{\mathbf{P}}$	$X_{\mathbf{P}}$	$X_{\mathbf{P}}$	$X_{\mathbf{P}}$	G	$X_{\mathbf{2P}}$
reg 2	$Z_{\mathbf{P}}$	$Z_{\mathbf{P}}$	$Z_{\mathbf{P}}$	E	E	$Z_{\mathbf{2P}}$
reg 3	$X_{\mathbf{Q}}$	–	D	$X_{\mathbf{P+Q}}$	$X_{\mathbf{P+Q}}$	$X_{\mathbf{P+Q}}$
reg 4	$Z_{\mathbf{Q}}$	–	$Z_{\mathbf{P+Q}}$	$Z_{\mathbf{P+Q}}$	$Z_{\mathbf{P+Q}}$	$Z_{\mathbf{P+Q}}$
reg 5	–	A	A	F	F	–
reg 6	–	B	–	–	H	–
reg 7	–	C	–	–	I	–
reg 8	$x_{\mathbf{G}}$	$x_{\mathbf{G}}$	$x_{\mathbf{G}}$	$x_{\mathbf{G}}$	$x_{\mathbf{G}}$	$x_{\mathbf{G}}$

Dans l'ensemble de l'algorithme, seuls ces 8 registres nécessitent donc une capacité en écriture et en lecture. Il est à noter que 2 accès en lecture sont nécessaires à chaque instant. Pour réaliser ces registres, il est possible d'utiliser les mémoires à double ports embarquées dans les FPGA. L'instanciation d'une mémoire à triple ports se fait alors par l'adjonction de 2 mémoires à double ports, le port d'écriture étant commun, alors que le port de lecture est spécifique à chacune. Néanmoins, le besoin en mémoire de ce type ne dépasse pas 576 bits (chaque variable locale étant nécessairement utilisée sur $\mathcal{B}$ et $\mathcal{B}'$, 16 registres de ce type sont nécessaires, d'au maximum 36 bits). Enfin, les blocs mémoires des FPGA utilisent des barres de registre sur le bus d'adressage et de la donnée lors d'une écriture, et pour une utilisation optimale (à vitesse maximale), il est recommandé d'utiliser une barre de registre en sortie de la mémoire en lecture. Or, il faut noter que le premier étage de pipeline de l'ALU est nécessairement un multiplieur, et qu'entre la sortie de la mémoire et le multiplieur, il existe un multiplexeur permettant de choisir X_0 provenant de la mémoire principale ou du bus (voir figure 2.3).

Par conséquent, il est possible au vu de ces contraintes d'utiliser un banc de registres utilisant la logique plutôt que le banc mémoire. L'écriture peut être implémentée sans attendre un cycle d'horloge, et permet d'implémenter dans le même étage de pipeline une addition modulo m_i et le stockage. Comme nous le verrons dans la section suivante, cela permet d'économiser un étage de pipeline, ce qui s'avère très utile pour limiter les temps d'attente de pipeline (notamment pour les petits niveaux de sécurité, où $n = 5$).

La figure 3.3 montre ainsi la structure générale de la mémoire, qui est partagée entre 16 registres et une mémoire accessible uniquement en lecture simple port. Ces ROMs, ainsi que les mémoire ROM_1 et ROM_2 incluent les précaluls nécessaires aux changements de bases au calcul modulo p du tableau 2.1. Par ailleurs, pour la réalisation des calculs sur courbe elliptique, les valeurs suivantes sont nécessaires (et spécifiques à la courbe) :

Algorithme	Nom	nombre	valeur	ROM
Courbe	$\tilde{a_4}$	$2n$	$\|\|a_4\mathcal{M}(\mathcal{B})\|_p\|_{m_i}$	principale
	$\tilde{a_6}$	$2n$	$\|\|a_6\mathcal{M}(\mathcal{B})\|_p\|_{m_i}$	principale
	$3p$	$2n$	$\|3p\|_{m_i}$	principale
	$-4\tilde{a_6}$	$2n$	$\|\|-4a_6\mathcal{M}(\mathcal{B})\|_p\|_{m_i}$	principale

3.3.2 Structure du pipeline

Comme indiqué dans la section 2.4.3, l'état interne de l'ALU de chaque Rower $U(t)$ doit respecter l'équation 2.6. Il convient d'abord de définir la profondeur maximale du pipeline. La contrainte la plus forte vient en fait des faibles niveaux de sécurité (160 bits) associés à de gros multiplieurs $r = 36$. Dans ce cas $n = 5$. Afin d'arriver à l'occupation maximale du pipeline, une taille de pipeline de 6 ne doit pas être dépassée (soit les opérandes X_0 et X_1 doivent être disponibles 6 cycles avant la mise à jour de $U(t)$). En effet, au delà, de trop nombreux cycles d'attente sont générés (l'effet de seuil est dû en grande partie au fait que l'algorithme 14 propose une utilisation de 6 cycles d'affilée du pipeline, ce qui permet une occupation maximale jusqu'à une profondeur de 6).

Les opérations décrites dans le tableau 3.4 ci-dessous permettent ainsi une réduction en 6 cycles, où chaque cycle n'est occupé que par une multiplication ou une addition modulaire.

Théorème 7 *Pour tout $\varepsilon < 2^{r/2}$, tout $\{X_0, X_1\} \in [0; 2^r]$, le pipeline décrit dans le tableau 3.4 permet le calcul de l'équation 2.6. Par ailleurs, les additions modulaires ont des opérandes inférieurs à $2^r - \varepsilon$ (à l'exception de l'étage 2).*

Preuve : On considère que l'état des accumulateurs au cycle $t-1$, $acc_1(t-1)+acc_2(t-1) = |U_{t-1}|_{2^r-\varepsilon}$. Soit $\theta = |X_0.X_1|_{2^r-\varepsilon}$. A l'étage 2, $MSB_1.2^r + LSB_1 = X_0 \times X_1$, donc $|MSB_1.\varepsilon + LSB_1|_{2^r-\varepsilon} = |\theta|_{2^r-\varepsilon}$. Ainsi $|MSB_2.2^r + LSB_2 + SR2|_{2^r-\varepsilon} = |\theta + C|_{2^r-\varepsilon}$. Ce qui implique que $|acc_1(t)+acc_2(t)|_{2^r-\varepsilon} = |MSB_2.\varepsilon+SR3|_{2^r-\varepsilon}$ ou $|acc_1(t)+acc_2(t)|_{2^r-\varepsilon} = |MSB_2.\varepsilon + SR3 + U(t-1)|_{2^r-\varepsilon}$, soit $|acc_1(t) + acc_2(t)|_{2^r-\varepsilon} = U(t)$. ε étant par ailleurs inférieur à $2^{r/2}$, la deuxième proposition est triviale à vérifier.

L'addition modulaire peut être réalisée de la manière décrite dans l'algorithme 19.

FIGURE 3.3 – Structure de Rower pour les courbes elliptiques

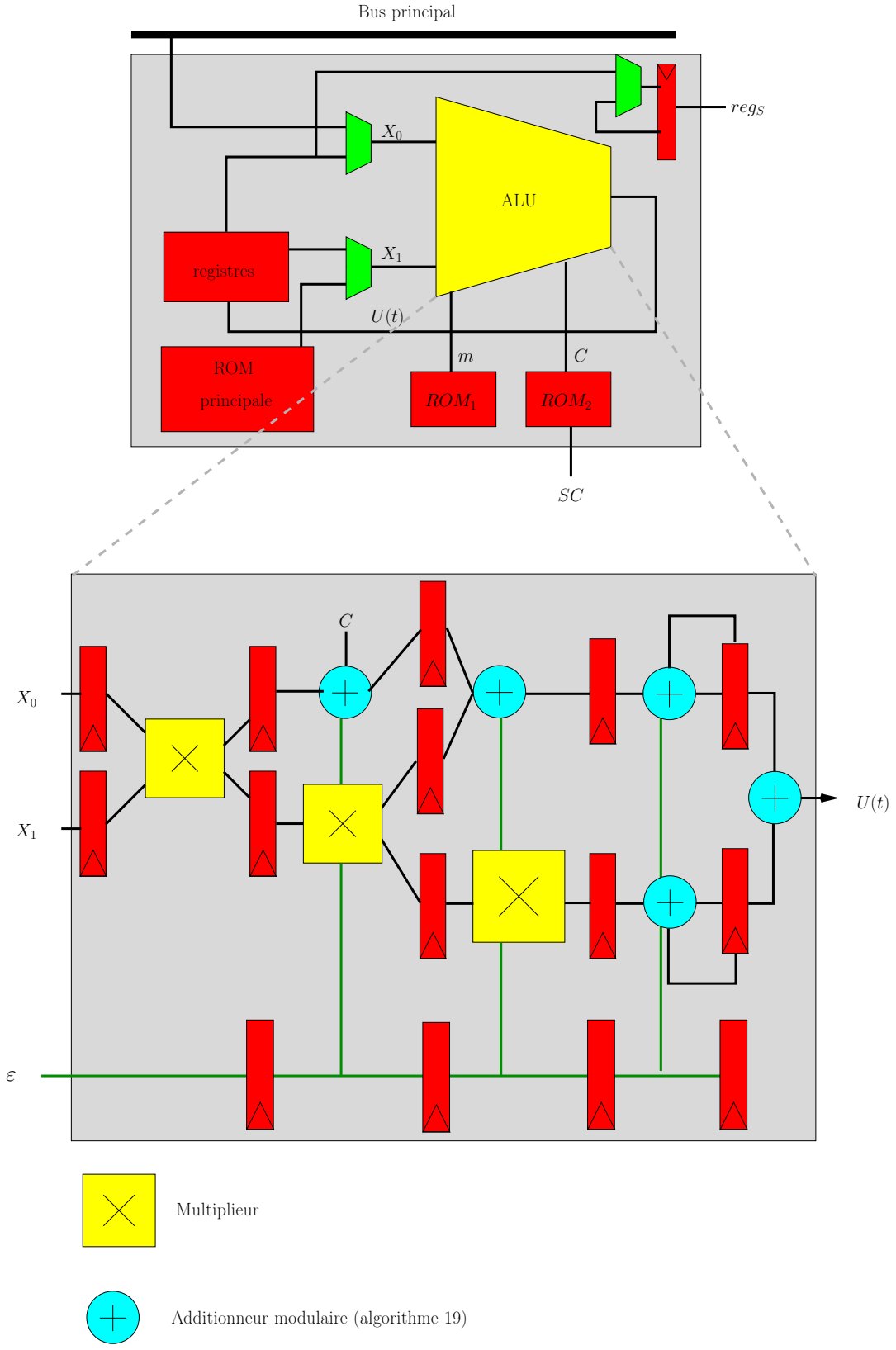


TABLE 3.4 – Structure du pipeline

étage 1	lecture
étage 2	$MSB_1 \| \| LSB_1 = X_0 \times X_1$
étage 3	$MSB_2 \| \| LSB_2 = MSB_1 \times \varepsilon$ et $SR2 = \|LSB1 + C\|_{2^r-\varepsilon}$
étage 4	$LSB3 = MSB_2 \times \epsilon_i$ et $SR3 = \|LSB2 + SR2\|_{2^r-\varepsilon}$
étage 5	($acc_1 = \|acc_1 + LSB3\|_{2^r-\varepsilon}$ et $acc_2 = \|acc_2 + SR3\|_{2^r-\varepsilon}$) ou ($acc_1 = LSB3$ et $acc_1 = SR3$)
étage 6	$out = \|acc_1 + acc_2\|_{2^r-\varepsilon}$ et écriture de *out*

Algorithme 19 : Additionneur modulaire

ENTRÉE(S): ε et $(x, y) \in [0; 2^r - \varepsilon]$
SORTIE(S): $|x + y|_{2^r-\varepsilon}$

1: $a = x + y$
2: $b = x$ xor y xor ε
3: $c =$ MAJ(x, y, ε)
4: $d||e = x + 2C$
5: **si** $d = 1$ **alors**
6: **retourne** e
7: **sinon**
8: **retourne** a
9: **fin si**

On peut noter que l'algorithme 19 permet de paralléliser simplement les 2 additions plutôt que de les réaliser en série, ce qui accélère le traitement (même si la consommation en portes est plus importante). Par ailleurs, si $2^{r+1} - \varepsilon \leq x + y < 2^{r+1}$ (ce qui peut être le cas à l'étage 2), l'algorithme 19 ne produit qu'un résulat compris entre $2^r - \varepsilon$ et 2^r. On peut aisément contourner ce problème en utilisant une base adaptée. En effet, les valeurs de C sont fixées par la base, et ne sont qu'au nombre de $2n$. En pratique, il est aisé de rejeter toutes les bases telles que $\exists i \in [1; n], C_{i,\mathcal{B}\to\mathcal{B}'} \geq 2m'_i - 2^r$ ou $\exists i \in [1; n], C_{i,\mathcal{B}'\to\mathcal{B}} \geq 2m_i - 2^r$. Une fois cette condition réalisée, l'algorithme 19 peut être utilisé pour toutes les additions de la figure 3.3. J'ai pu noter durant cette étude que le circuit généré par l'algorithme 19 a un chemin critique équivalent à la multiplication, et par conséquent ne retarde pas le Cox-Rower.

Dans ce travail, le même pipeline est utilisé pour toutes les tailles de courbes, même si on peut noter que cela n'est pas obligatoire. En effet, pour des tailles plus grandes de n, il est possible d'ajouter quelques étages, sans créer de cycles d'attente. Cette démarche peut nécessiter des étapes supplémentaires, mais permet aussi l'utilisation des mémoires à double ports des FPGA. Ce compromis peut être fait en fonction de la technologie visée.

3.3.3 Sécurité

Les implémentations des courbes elliptiques sont aussi naturellement sensibles aux attaques par canaux auxiliaires et aux perturbations décrites dans la section 1.3.

Les SPA, DPA ou attaques template s'appliquent de manière évidente sur des implémentations non protégées. On se place dans le cas d'une implémentation de $k\mathbf{G}$ avec k secret, et $\mathbf{G}$ pouvant être choisi par l'attaquant. Pour parer la SPA, l'échelle de Montgomery peut être utilisée, ou bien les formules unifiées d'addition et de doublement de [32]. Dans notre cas, l'échelle de Montgomery donne des meilleurs résultats [13]. Pour parer la DPA ainsi que les attaques template [38], Coron propose 3 contre-mesures à un coût raisonnable :

- la première consiste à masquer les coordonnées projectives du point $\mathbf{G}$ au début du calcul, en multipliant $X_{\mathbf{G}}$ et $Z_{\mathbf{G}}$ par une même valeur aléatoire. Le coût de cette contre-mesure est quasiment nul. En revanche, l'attaque de Goubin [58] utilise le fait que l'attaquant peut choisir $X_{\mathbf{G}} = 0$ dans certains cas, et cette valeur ne peut pas être masquée multiplicativement, ce qui peut être détecté par l'attaquant. Une autre attaque d'Itoh et al. [72] quand k est un nonce (dans le cas des signatures basées sur El-Gamal par exemple) utilise l'"address bit DPA" de [95] spécifiquement dans le cas des courbes.
- la deuxième contre-mesure consiste à masquer l'exposant : si α est un aléa, k peut être remplacé par $k + \alpha\#\mathcal{C}$ pour obtenir le même résultat. Cette contre-mesure peut être attaquée par les attaques template de [49, 68].
- la troisième est le masquage du point. $\mathbf{G}$ est remplacé par $\mathbf{G} + \mathbf{R}$ où $\mathbf{R}$ est un point aléatoire. $k\mathbf{R}$ est par ailleurs aussi calculé et soustrait à la fin du calcul. Cette contre-mesure est assez onéreuse puisqu'elle multiplie par 2 le temps de calcul.

Jusqu'à présent, seules des attaques très puissantes comme l'"address bit SPA" permettent d'obtenir des résultats quand les 2 premières contre-mesures sont mises en œuvre (chacune neutralisant les attaques de l'autre). On peut constater que ces contre-mesures sont très aisées à mettre en œuvre sur notre Cox-Rower, et le coût en efficacité associé est égal à $log_2(\alpha)/log_2(p)$. La troisième contre-mesure peut, elle aussi, être implémentée, mais nécessite des registres supplémentaires pour conserver les valeurs $X_{k(\mathbf{G}+\mathbf{R})}$, $Y_{k(\mathbf{G}+\mathbf{R})}$, $Z_{k(\mathbf{G}+\mathbf{R})}$ pendant le calcul de $k\mathbf{R}$, afin de mutualiser l'inversion finale, pour un coût final de plus de 80% du temps total de l'exponentiation.

Les attaques par perturbation peuvent permettre d'envoyer les points manipulés sur des courbes elliptiques différentes où le logarithme discret est facile. Par exemple, [48] propose de réaliser une faute sur la valeur initiale de $X_{\mathbf{G}}$, pour que celui-ci ne soit pas sur la courbe. Il est aisé de démontrer que dans ce cas le calcul s'effectuera suivant les formules du tableau 3.2 sur la courbe d'équation

$$y^2 = x^3 + a_4 x - a_6 \tag{3.9}$$

qui n'est autre que la courbe tordue de $\mathcal{C}$ et qui possède $2p - \#(\mathcal{C})$ points sur $\mathbb{F}_p$. Si cette valeur est friable, alors l'attaque permet de retrouver k (c'est en outre le cas d'un certain nombre de courbes normalisées par [100]).

Un moyen simple de pallier ce type d'attaque est naturellement de vérifier si le point calculé est sur la courbe. Cette opération est toujours possible et très efficicace. Elle peut être réalisée par un Cox-Rower, qui transmettra le résulat de $(y^2_{k\mathbb{G}} - x^3_{k\mathbb{G}} - a_4 x_{k\mathbb{G}} - a_6)/p$ pour être comparé à O,1 ou 2.

Enfin, les "safe error" peuvent être utilisées contre les implémentations sur courbes elliptiques, notamment les "C-safe error", perturbant les opérations "dummy" insérées pour se protéger de la SPA, ou les "M-safe error", qui attaquent la mémoire quand l'utilisation

de celle-ci dépend de la clé [122]. L'utilisation de l'échelle de Montgomery, en respectant l'ordre des opérations (doublement, puis addition), permet de réduire cette vulnérabilité.

En conclusion, notre architecture propose l'implémentation simple des trois contre-mesures présentées dans ce chapitre qui permet de parer les attaques connues où le modèle d'attaquant est assez faible, pour un prix très modéré sur l'implémentation. Il faut noter que dans ce domaine, il y a une grande différence avec [108] ou [63] dont l'objectif est la performance sur FPGA, et qui utilisent des méthodes NAF, plus performantes en représentation binaire, mais qui nécessitent l'insertion d'opérations inutiles (dummy) pour être protégées contre la DPA, et deviennent finalement sensibles aux "safe error".

3.3.4 Conclusion

La configuration du pipeline proposée ci-dessus permet un taux d'occupation globale du pipeline de plus de 90% sur l'ensemble de la multiplication scalaire. L'ensemble des cycles inoccupés se trouvent dans l'inversion modulaire (qui est une exponentiation par $p-2$, faute de pouvoir réaliser autrement l'inversion modulaire dans un Cox-Rower), dans la transformation RNS $\rightarrow$ binaire et binaire $\rightarrow$ RNS. Par ailleurs, le taux d'occupation évolue en fonction de la taille de p et de n, le pire cas étant toujours pour les petites courbes et $n = 5$. L'étude a été réalisée pour optimiser la vitesse du circuit sur Stratix. Mais le portage sur une autre technologie consiste uniquement au dimensionnement de n et à la restructuration de l'architecture du pipeline.

3.4 Performances et comparaisons

Dans cette section, je donne les résultats d'implémentation sur différentes technologies de FPGA, et je les compare à d'autres résultats obtenus dans la littérature ouverte.

3.4.1 Résultat d'implémentation pour l'ensemble des niveaux de sécurité

La technologie cible choisie est la famille Altera Stratix. Le choix s'est porté sur Altera plutôt que sur Xilinx du fait de la disponibilité de la chaîne de compilation Altera Quartus II pendant l'étude. Néanmoins, des études indépendantes comme [102] démontrent que l'impact d'un tel choix est faible. Parmi l'ensemble des technologies Altera disponibles, j'ai choisi 2 nœuds spécifiques à des fins de comparaison. La première est la famille Altera Stratix, le deuxième est un Stratix II (voir section 1.1.3).

Toutes les courbes choisies le sont de manière aléatoire aux tailles suivantes : 160, 192, 256, 384 and 512 bits. p, a_4 and a_6 sont ainsi choisis avec aucune autre contrainte que C_{p,a_4,a_6} soit lisse, et que $\#C_{p,a_4,a_6}$ soit premier. Les résultats indiqués ci-dessous ne dépendent pas de ces valeurs mais sont déterminés par $log_2(p)$. Le nombre de Rower est établi à n ($\eta = 1$). Les valeurs de r choisies dans le tableau 3.5 apportent le plus d'efficacité.

Pour chaque génération de bitstream je donne la référence exacte du circuit choisi. La fréquence maximale atteignable ainsi que le temps de calcul complet pour une multiplication scalaire $k\mathbf{G}$ sont donnés. k est choisi de la taille de p, et donc de $\#C_{p,a_4,a_6}$. L'inversion finale, le calcul de $y_{[k]\mathbf{G}}$, la transformation RNS $\rightarrow$ binaire et binaire $\rightarrow$ RNS pour les 2 coordonnées affines de chaque point sont inclus dans ce résultat. Enfin, l'occupation des ressources dans le FPGA est donnée en fonction de la technologie. Le Stratix est composé de "Logic Element" (LE), qui sont de simples LUT comme indiqués dans la figure 1.4, auxquelles sont adjointes des capacités d'addition. Les Stratix II sont réalisés à partir d' Altera

Logic Module(ALM) de la figure 1.5, équivalent du slice du Virtex IV. Le nombre de blocs DSP (multiplieurs) est aussi donné. Altera donne le taux d'occupation de DSP en nombre de multiplieurs 9×9 occupés. En effet les blocs DSP du Stratix et Stratix II peuvent être configurés comme un multiplieur 36×36, 2 multipleurs 18×18 ou 8 multiplieurs 9×9. Sur Stratix II, ces multiplieurs ont une capacité d'accumulation que je n'utilise pas dans ces travaux.

TABLE 3.5 – Résultats détaillés sur Altera Stratix et Stratix II

Famille	$log_2(p)$	modèle	n	r	taille	DSP	freq. (MHz)	latence
Stratix	160	EP1S20F484C5	5	34	11431 LE	74	92.6	0.57 ms
	192	EP1S30F780C5	6	33	12480 LE	80	89.6	0.72 ms
	256	EP1S60F780C5	8	33	16200 LE	125	90.7	1.17 ms
	384	EP1S80F1020C5	11	36	25279 LE	176	90.0	2.25 ms
	512 [1]	EP1S80F1020C5	15	35	48305 LE	176	79.6	4.03 ms
Stratix II	160	EP2S30F484C3	5	34	5896 ALM	74	165.5	0.32 ms
	192	EP2S30F484C3	6	33	6203 ALM	92	160.5	0.44 ms
	256	EP2S30F484C3	8	33	9177 ALM	96	157.2	0.68 ms
	384	EP2S60F484C3	11	36	12958 ALM	177	150.9	1.35 ms
	512	EP2S60F484C3	15	35	17017 ALM	244	144.9	2.23 ms

Les résultats obtenus dans le tableau 3.5 apportent des éclairages sur le comportement de l'architecture Cox-Rower à différents niveaux de configuration. Tout d'abord, on peut constater que la fréquence d'utilisation ne décroit que très lentement avec la taille du circuit. Ce résultat est prévisible dans la mesure où il n'existe aucun chemin dont la taille dépende de n à l'exception du Mux et du bus principal. Dans les circuits proposés, il n'existe pas d'élément constituant à coup sûr le chemin critique. Pour certains circuits le chemin critique est dans le séquenceur, pour d'autres il est dans les Rowers. Aucune instruction n'a été donnée au logiciel de génération de bitstream à l'exception de la recherche d'une fréquence de fonctionnement maximale.

Comme nous avons vu dans la section 3.3, je n'ai pas utilisé de blocs de mémoire dans ce circuit. Même si les blocs de RAM sont intégrés dans les FPGA, ce choix est motivé par la volonté de limiter le nombre de niveaux de pipeline. Comme nous le verrons dans les chapitres 4 et 5, ce choix multiplie par 2 la quantité de logique utilisée, mais libère complètement les ressources de mémoire. La famille Stratix manque de blocs DSP même dans les plus gros Stratix pour le niveau 512 bit, tandis que la famille Stratix II présente largement assez de ressources pour tous les niveaux de sécurité proposés dans ces travaux. A titre d'exemple, Le EP2S60F484C3 est une matrice de taille moyenne dans la famille Stratix II.

3.4.2 Comparaisons

La contribution [41] propose une synthèse des circuits publiés avant 2006 qui implémentent la multiplication scalaire sur courbes elliptiques et qui visent la haute performance.

1. L' EP1S80 ne possède pas suffisemment de blocs DSP, les multiplieurs sont mappés dans les Logic Elements, et la fréquence diminue.

Les implémentations peuvent être divisées en 2 catégories : celles qui supportent les courbes elliptiques sur les corps de caractéristique 2, et celles sur les corps de type $\mathbb{F}_p$ avec p premier. Le premier groupe donne le meilleur ratio vitesse sur sécurité. La raison essentielle est la vitesse de l'arithmétique des extensions finies de $\mathbb{F}_2$ en matériel (aucune propagation de retenue n'est nécessaire, et pour certaines courbes comme les courbes de Koblitz, l'étape de doublement peut être astucieusement remplacée par un Frobenius, qui est une application linéaire). Les implémentations à l'état de l'art présentent, par exemple, une latence de 20 μ pour une sécurité de 2^{80} comme par exemple [75]. Néanmoins, les implémentations matérielles sur $\mathbb{F}_p$ restent intéressantes à étudier, pour 2 raisons :

- il est admis dans la communauté que le logarithme discret est plus facile sur les corps de caractéristique 2 que sur $\mathbb{F}_p$, même si aucun algorithme ne permet de traiter le cas général plus efficacement que les courbes génériques dans les 2 cas. Une des raisons principale de cette conjecture est que la fonction carré dans $\mathbb{F}_2$ linéaire (il s'agit en effet du Frobenius).
- jusqu'à l'apparition du SSE 4.3 l'implémentation des courbes elliptiques définies sur $\mathbb{F}_2$ était peu efficace sur les architectures x86. En revanche, les courbes elliptiques définies sur $\mathbb{F}_p$ sont implémentées de manière très efficace sur ces architectures, (la meilleure implémentation à ce jour est proposée dans [90], ou même sur les architectures de type GPGPU (comme le processeur CELL dans [117]).

Il faut noter aussi que certaines architectures proposent le support des 2 types de corps [109].

Parmi les implémentations comparables de la littérature (implémentation sur FPGA de la multiplication scalaire sur courbes définies sur $\mathbb{F}_p$ avec comme objectif de réduire la latence), je compare cette implémentation avec 3 publications significatives de la littérature :

- le premier [111] est une autre implémentation FPGA basée sur le RNS. Dans cette contribution, les auteurs utilisent un shéma de Horner pour réaliser la transformation RNS $\rightarrow$ binaire et binaire $\rightarrow$ RNS à chaque addition ou doublement de point. C'est à ma connaissance la seule implémentation utilisant le RNS pour les courbes elliptiques. Les auteurs proposent une instanciation sur ASIC et sur FPGA (Virtex).
- la seconde est décrite dans [94], et utilise une approche binaire parallèle (l'algorithme utilisé est un algorithme de Montgomery (algorithme 2)), utilisant des multiplieurs en parallèle. Cette contribution se distingue par la recherche de parallélisme dans l'algorithme Montgomery pour dessiner un pipeline réalisant l'étage 4 de la multiplication de Montgomery (algorithme 4), et par l'utilisation de "carry-look adder" pour les additions de grands nombres. La technologie visée est le Virtex 2-pro de Xilinx. La multiplication scalaire est basée sur le recodage NAF. C'est à ma connaissance l'implémentation la plus rapide de la multiplications scalaire sur courbes elliptiques génériques (i.e, p n'a pas de caractéristiques particulières). Cette implémentation dépasse l'ensemble de celles données dans le survey [41].
- la troisième contribution est celle de [63]. Le circuit est spécifique aux courbes NIST, p est donc un pseudo-Mersenne. Cela limite la complexité de la réduction modulaire par 2. L'idée principale est l'utilisation d'un circuit à 2 domaines d'horloge, les multiplieurs pouvant fonctionner à leur vitesse maximale, soit 500 MHz. Il s'agit de l'implémentation FPGA la plus rapide existante pour la multiplication scalaire sur $\mathbb{F}_p$, mais avec des restrictions sur p et sur sa taille.

Comme le montre le tableau 3.6, notre implémentation est largement plus rapide et petite que celle de [111]. L'architecture de [94] présente un meilleur compromis temps surface. Néanmoins, le temps de calcul est plus faible sur notre technologie. Par ailleurs, l'utilisation

TABLE 3.6 – Comparaison de notre implémentation avec d'autres contributions

contribution	courbe	famille FPGA	modèle FPGA	taille	freq.(MHz)	latence
Ce travail	160	Stratix	EP1S20F484C5	11431 LE	92.6	0.57 ms
	256	Stratix	EP1S60F780C5	16200 LE	90.7	1.17 ms
	160	Stratix II	EP2S30F484C3	6203 ALM	165.5	0.32 ms
	256	Stratix II	EP2S30F484C3	9177 ALM	157.2	0.68 ms
[111]	160	Virtex	XCV1000E-8	21000 LUT	58	1.77 ms
	256	Virtex	XCV1000E-8	36000 LUT	39.7	3.95 ms
[94]	160	Virtex II-pro	XC2VP30	2171 sl.	72	1 ms
	256	Virtex II-pro	XC2VP30	3529 sl.	67	2.27 ms
[63]	224 (NIST)	Virtex 4	XC4VFX12	1580 sl.	487	0.36 ms
	256 (NIST)	Virtex 4	XC4VFX12	1715 sl.	490	0.49 ms

de banc de registre plutôt que de RAM embarqués fait descendre le compromis temps surface. Enfin, l'implémentation de [94] n'est pas résistante aux attaques side channel, tandis que celle présentée dans ce chapitre l'est nativement.

L'architecture décrite dans [63] est plus rapide et présente un meilleur compromis temps surface (si l'on convient qu'un ALM de Stratix II et un slice de Virtex IV sont équivalents, et si l'on considère uniquement les slices). L'architecture de [63] calcule $[k]\mathbf{P}$ sur les courbes NIST. On peut considérer que le facteur de réduction de la complexité dans le cas de pseudo-Mersenne est d'environ de $1,68$ dans la mesure où la réduction fainéante est utilisée. Ainsi, nous pouvons constater que les résultats que j'ai obtenus sont compétitifs si l'on regarde la latence à un niveau 224 bits, et sont meilleurs pour 256 bits. Par ailleurs, l'implémentation de [63] nécessite l'utilisation d'une fréquence de 250 et 500 MHz, ce qui est particulièrement rapide et peut constituer un obstacle pour une intégration industrielle ou pour une implémentation ASIC (où l'utilisation de multiplieurs 2 fois plus rapides que le reste du circuit est impossible). Bien sûr ces comparaisons ne peuvent pas être précises dans la mesure où les technologies visées, les courbes, et les objectifs de sécurité ne sont pas les mêmes, mais l'objectif est de démontrer dans ce travail que le RNS est une alternative très compétitive pour les courbes définies sur $\mathbb{F}_p$, vue la capacité à s'adapter sur de nombreuses technologies et à différents niveaux de sécurité. La compétitivité, par ailleurs, s'améliore avec le niveau de sécurité.

3.5 Conclusion

Dans ce chapitre, j'ai démontré que l'architecture Cox-Rower permettait l'implémentation efficace de la multiplication scalaire sur courbes elliptiques et ce à tous les niveaux de sécurité classiquement utilisés en cryptographie.

Chapitre 4

Implémentation efficace de couplage sur les courbes Barreto-Naehrig

Sommaire

4.1 Introduction 62
4.2 Couplage sur les courbes elliptiques 63
4.2.1 Définition et motivation 63
4.2.2 Fonctions de Miller et couplage de Tate 65
4.2.3 Couplages de Ate et Optimal Ate 66
4.2.4 Les courbes Barreto-Naehrig 66
4.2.5 Choix de courbes et extension de corps 68
4.3 Calcul de couplage avec un Cox-Rower 68
4.3.1 Paramétrisation du Cox-Rower pour le couplage 69
4.3.2 Architecture du pipeline 69
4.4 Arithmétique sur les extensions de $\mathbb{F}_p$ 71
4.4.1 Arithmétique sur $\mathbb{F}_{p^2}$: retour à la méthode naïve 71
4.4.2 Arithmétique sur $\mathbb{F}_{p^{12}}$, étude de l'apport des techniques d'interpolation 71
4.4.3 Inversion sur $\mathbb{F}_{p^{12}}$ 73
4.4.4 Séquencement de plus haut niveau 73
4.4.5 Contrôle des variables locales 75
4.5 Résultats d'implémentation, analyse et comparaison 76
4.5.1 Taille du circuit 76
4.5.2 Latence d'un couplage 76
4.5.3 Comparaisons et discussions 77
4.6 Conclusion 77
4.7 Annexe A : circuit optimisé pour la courbe BN_{126} sur FPGA Virtex 6 79

4.1 Introduction

Les couplages sur courbes elliptiques ont été introduits en cryptographie dans le milieu des années 1990, tout d'abord dans un objectif de cryptanalyse. En effet, il permettent de remplacer la recherche du logarithme discret sur une courbe elliptique où il n'existe à ce jour aucune attaque sous-exponentielle par le logarithme discret sur un corps fini, où

existent des algorithmes sous-exponentiels, comme les algorithmes de crible FFS [51, 93]. Pour certaines courbes, il s'avère que le problème est plus facile à résoudre sur le corps fini que sur la courbe, et dégrade donc la sécurité générale de la courbe. En 2000, Joux eut l'idée d'utiliser les couplages à des fins de construction et non de cryptanalyse : l'échange Diffie Hellman tripartite [76]. Depuis, les couplages sont devenus un outil pour le développement de familles de protocoles, dont les contributions les plus connues sont la cryptographie basée sur l'identité [28] ou les signatures courtes [29]. Ainsi, le couplage est devenu populaire, et la capacité de construire des courbes bénéficiant de bonnes propriétés (dites courbes "pairing friendly"), au bon niveau de sécurité, et de les calculer efficacement est devenue un sujet de recherche important, comme le démontrent les nombreuses publications sur ce sujet [67, 118, 98, 24, 12, 45]. Dans ce chapitre, je m'intéresse à l'implémentation matérielle de calculs de couplages sur courbes elliptiques à un niveau égal ou supérieur à 128 bits. Parmi l'ensemble des choix possibles de courbes, j'utilise les courbes de Barreto-Naehrig, une famille paramétrée de courbes qui présentent de très bonnes caractéristiques à ce niveau de sécurité. En effet, la découverte récente de ces courbes et l'utilisation d'un couplage efficace sur celles-ci appelé "optimal pairing" [118] ont permis d'obtenir des records de vitesse significatifs en logiciel [12]. Dans ce chapitre, je propose de combiner ces améliorations avec le coprocesseur décrit dans le chapitre 3, pour obtenir l'implémentation de couplages sur courbes elliptiques la plus rapide obtenue à 128 bit de sécurité au moment de sa publication et la première implémentation matérielle d'un couplage à 192 bits de sécurité [1].

Le reste du chapitre est structuré de la manière suivante. La section 4.2 réalise un état de l'art succinct des couplages, de leur utilisation et de leur implémentation. Je réfère le lecteur à [77] pour un état de l'art plus complet sur le sujet. La section 4.3 décrit l'implémentation qui m'a permis d'obtenir les résultats sur le couplage, une implémentation du Cox-Rower avec un pipeline spécifiquement étudié pour le couplage. La section 4.4 décrit le séquencement du couplage, en proposant des choix algorithmiques à tous les étages du couplage. Enfin, la section 4.5 donne les résultats d'implémentation, et les compare avec le reste de la littérature. En annexe, une description brève du travail de mes coauteurs de [35] est donnée, pour une implémentation spécifique de couplages encore plus rapide.

4.2 Couplage sur les courbes elliptiques

4.2.1 Définition et motivation

Commençons par définir ce qu'est un couplage.

Définition 10 *Soient* $(\mathbb{G}_1, +)$, $(\mathbb{G}_2, +)$ *et* $(\mathbb{H}, \times)$ *trois groupes abéliens. Un couplage de* $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{H}$ *est une fonction bilinéaire et non dégénérée.*

Soit e le couplage, et les éléments neutres $\mathcal{O}_{\mathbb{G}_1}$, $\mathcal{O}_{\mathbb{G}_2}$ et $\mathbf{1}_{\mathbb{H}}$, la bilinéarité s'entend par

$$e(\mathbf{P}+\mathbf{Q},\mathbf{R}) = e(\mathbf{P},\mathbf{R}) \times e(\mathbf{Q},\mathbf{R}) \text{ et } e(\mathbf{P},\mathbf{R}+\mathbf{S}) = e(\mathbf{P},\mathbf{R}) \times e(\mathbf{P},\mathbf{S})$$

et la non dégénérescence par

$$(\forall \mathbf{Q} \in \mathbb{G}_2 \; e(\mathbf{P},\mathbf{Q}) = \mathbf{1}_{\mathbb{H}}) \Rightarrow \mathbf{P} = \mathcal{O}_{\mathbb{G}_1}$$
$$(\forall \mathbf{P} \in \mathbb{G}_1 \; e(\mathbf{P},\mathbf{Q}) = \mathbf{1}_{\mathbb{H}}) \Rightarrow \mathbf{Q} = \mathcal{O}_{\mathbb{G}_2}$$

1. Ce chapitre est issu de la contribution [35], publication issue d'une fusion des prépublications [44] que j'ai réalisé en collaboration avec Sylvain Duquesne et [121]. Ma contribution à [35] durant cette thèse est la définition et le développement de l'architecture générique ainsi que des améliorations sur l'architecture optimisée permettant un gain de 18% entre [121] et la publication finale [35]. Le cœur de ce chapitre ne traite que de la description de l'architecture générique, le lecteur trouvera en annexe 4.7 l'optimisation de mes coauteurs exposée dans [35]

Pour des applications concrètes en cryptographie, les logarithmes discrets sur les groupes $\mathbb{G}_1$,$\mathbb{G}_2$ et $\mathbb{H}$ doivent être difficiles (la sécurité globale du cryptosystème utilisant les couplage sera celle du groupe le plus faible), et le calcul du couplage doit être (aussi) efficace (que possible).

Si toutes ces conditions sont réunies, il est alors possible de réaliser des fonctions non accessibles en utilisant des primitives classiques comme le RSA et les courbes elliptiques. Un exemple intéressant est le chiffrement basé sur l'identité, où chaque destinataire de message va choisir sa clé publique plutôt que sa clé privée. Ainsi, cette clé publique pourra être simplement son nom ou son adresse mail. Un tel mécanisme a pour avantage de simplifier grandement l'infrastructure de gestion des clés, puisqu'il n'est plus nécessaire de maintenir des certificats.

Par exemple, le système de chiffrement basé sur l'identité de Boneh et Franklin [28] met en œuvre 3 acteurs : une autorité en charge de délivrer les clés privées notée A, un émetteur E et un destinataire D. Pour simplifier les notations, je considère que $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ (c'est d'ailleurs le cas dans la proposition de [28]). On note $\mathbf{G}$ un élément générateur de $\mathbb{G}$. Ce cryptosystème nécessite par ailleurs 2 fonctions de hachage H_1 de $\{0,1\}^*$ vers $\mathbb{G}$ et H_2 de $\mathbb{H}$ vers $\{0,1\}^*$.

Le système de chiffrement nécessite 4 étapes :

Génération de la clé d'autorité : A choisit une clé privée sk_A et calcule $\mathbf{P}k_A = k\mathbf{G}$. Il publie $\mathbf{P}k_A$.

Distribution de la clé privée : D choisit une clé publique pk_D et calcule $H_1(pk_D) \in \mathbb{G}$ qu'il transmet à A, en retour A transmet $\mathbf{S}k_D = sk_A H_1(pk_D)$ via un canal sécurisé à A. D publie pk_D.[2]

Chiffrement d'un message m : E choisit r un aléa, calcule les valeurs suivantes
- $g = e(H_1(pk_D), \mathbf{P}k_A)$
- $\mathbf{Q} = r\mathbf{G}$
- $c = m \oplus H_2(g^r)$

Il publie $(\mathbf{Q}, c)$.

Déchiffrement d'un message (u, v) : D calcule $m' = v \oplus H_2(e(u, \mathbf{S}k_D))$ pour obtenir le message initial. Si ce message a été forgé en respectant la méthode de chiffrement décrite ci-dessus, $m' = m$.

Il existe de nombreux schémas de ce type à utiliser des couplages. Parmi eux, on peut citer le Diffie-Hellman tripartite en 1 passe [76], des mécanismes de signature courte [29], les mécanismes de signature basés sur l'identité [123, 66] ou des mécanismes de Broadcast [42, 91].

Nous allons voir maintenant que les courbes elliptiques proposent des couplages utilisables en cryptographie.

2. Cette phase est réalisée soit avant le chiffrement de message, auquel cas le chiffrement de message est réalisé avec pk_D de la manière décrite ci dessus, soit après le chiffrement, ce qui signifie que c'est l'émetteur qui choisit la clé publique. Dans ce dernier cas l'émetteur publie aussi cette clé, et le destinataire se charge de récupérer auprès de A la clé privée correspondante. Dans les deux cas, cette clé publique peut être choisie arbitrairement.

4.2.2 Fonctions de Miller et couplage de Tate

Je rappelle que $\mathbb{F}_q$ est un corps fini de caractéristique p, $\overline{\mathbb{F}_q}$ sa clôture algébrique, et $\mathcal{C}$ une courbe elliptique lisse sur $\mathbb{F}_q$. Soit ℓ un entier premier divisant $\#(\mathcal{C}/\mathbb{F}_q)$, et soit k le plus petit entier tel que ℓ divise $p^k - 1$.

Définition 11 *La ℓ-torsion de $\mathcal{C}$ est l'ensemble des points $\mathbf{P} \in \mathbb{P}^2(\overline{\mathbb{F}_q})$ tel que $\ell\mathbf{P} = \mathcal{O}$. Elle est notée $\mathcal{C}[\ell]$.*

Théorème 8 *Tous les membres de la ℓ-torsion sont définis sur $\mathbb{P}^2({\mathbb{F}_q}^k)$ et définissent un sous groupe isomorphe à $\mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$.*

Ce théorème est démontré dans [115] (chapitre 3, section 6.4).

Théorème 9 *Soit $\mathbf{P} \in C[\ell]$. Le diviseur $\ell(\mathbf{P}) - \ell(\mathcal{O})$ est un diviseur principal.*

La preuve est une preuve constructive puisqu'elle donne aussi l'algorithme pour calculer la fonction rationnelle correspondante. On utilise pour cela les fonctions de Miller d'un entier naturel n et d'un point $\mathbf{P}$ définies par récurrence

$$\begin{array}{rcl} f_{1,\mathbf{P}} & = & 1 \\ f_{a+b,\mathbf{P}} & = & f_{a,\mathbf{P}} f_{b,\mathbf{P}} \frac{g_{(a\mathbf{P},b\mathbf{P})}}{h_{((a+b)\mathbf{P})}} \\ f_{-a,\mathbf{P}} & = & \frac{1}{f_{a,\mathbf{P}} \times h_{(a\mathbf{P})}} \end{array}$$

où $g_{(\mathbf{Q},\mathbf{R})}$ est l'équation de la droite passant par $\mathbf{Q}$ et $\mathbf{R}$, tandis que $h_{(\mathbf{S})}$ est l'équation de la verticale passant par $\mathbf{S}$.

Il est aisé de démontrer que les diviseurs des fonctions de Miller sont égaux à

$$f_{n,\mathbf{P}} = n(\mathbf{P}) - (n\mathbf{P}) - (n-1)(\mathcal{O}). \tag{4.1}$$

On peut constater que si $\mathbf{P} \in \mathcal{C}[\ell]$, alors $f_{\ell,\mathbf{P}}$ est le diviseur recherché.

Ainsi, l'algorithme 20, appelé algorithme de Miller, permet de calculer $f_{n,\mathbf{P}}$ quelque soit $\mathbf{P}$, en utilisant un algorithme de doublement et addition classique agrémenté de la mise à jour d'un résultat intermédiaire avec le calcul des équations des lignes g et h. La justesse de l'algorithme se démontre aisément grâce aux formules de récurrence ci-dessus.

Algorithme 20 : Algorithme de Miller

Entrée(s): $\mathbf{P} \in \mathcal{C}, n \in \mathbb{N}$
Sortie(s): $f_{n,\mathbf{P}}$

1: $f \leftarrow 1$, $\mathbf{R} \leftarrow \mathbf{P}$
for i *de* $\lfloor \log_2(n) \rfloor - 2$ *à* 0 **do**
2: $f \leftarrow f^2$
3: $f \leftarrow f \times g_{(\mathbf{R},2\mathbf{R})}$
4: $f \leftarrow f / h_{(\mathbf{R})}$
5: $\mathbf{R} \leftarrow 2\mathbf{R}$
6: **if** $n_i = 1$ **then**
7: $f \leftarrow f \times g_{(\mathbf{R},2\mathbf{P})}$
8: $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{P}$
9: $f \leftarrow f / h_{(\mathbf{R})}$
endif
endfor
10: **retourne** f

Pour toute fonction rationnelle $f \in \overline{\mathbb{F}_q}(\mathcal{C})$ telles que $\mathbf{P}$ ne soit pas un pôle de f, $f(\mathbf{P})$ est la valeur de f au point $\mathbf{P}$.

Théorème 10 *La fonction suivante :*

$$e_T \left| \begin{array}{rcl} \mathcal{C}[\overline{\mathbb{F}_q}][\ell] \times \frac{\mathcal{C}[\overline{\mathbb{F}_q}]}{\ell\mathcal{C}[\overline{\mathbb{F}_q}]} & \rightarrow & \overline{\mathbb{F}_q}[\ell] \\ (\mathbf{P}, \mathbf{Q}) & \rightarrow & f_{\ell,\mathbf{P}}(\mathbf{Q})^{\frac{q^k-1}{\ell}} \end{array} \right.$$

est un couplage appelé couplage de Tate réduit [18].

4.2.3 Couplages de Ate et Optimal Ate

Par la suite $q = p$.

Dans la contribution [67] les auteurs étudient l'action du Frobenius sur le couplage de Tate, et arrivent à la construction d'un nouveau couplage pour les courbes en grande caractéristique appelé couplage de Ate. L'idée utilisée dans cette contribution adapte les arguments qui ont permis de découvrir le couplage η_T en caractéristique 2 et 3, pour le généraliser en grande caractéristique. En effet, les auteurs constatent que $\mathcal{C}(\mathbb{F}_{p^k})[\ell]$ est stable par le Frobenius π. Ainsi, π possède 2 valeurs propres sur $\mathcal{C}(\mathbb{F}_{p^k})[\ell]$: 1 (dont le sous-espace propre correspondant est le groupe $\mathcal{C}(\mathbb{F}_p)[\ell]$) et p.

Le couplage de Tate peut naturellement être réécrit en se restreignant à ces deux sous-espaces propres,

$$e_{T'} \left| \begin{array}{rcl} \mathcal{C}(\mathbb{F}_p)[\ell] \times \mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p) & \rightarrow & \mathbb{F}_{p^k}[\ell] \\ (\mathbf{P}, \mathbf{Q}) & \rightarrow & f_{\ell,\mathbf{P}}(\mathbf{Q})^{\frac{p^k-1}{\ell}} \end{array} \right.$$

ce qui permet de limiter la taille du corps $\mathbb{G}_1$ à la taille de $\mathbb{F}_p$. Via plusieurs manipulations successives sur les fonctions de Miller, les auteurs de [67] obtiennent le couplage suivant :

$$e_A \left| \begin{array}{rcl} \mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p) \times \mathcal{C}(\mathbb{F}_p)[\ell] & \rightarrow & \overline{\mathbb{F}_p}[\ell] \\ (\mathbf{Q}, \mathbf{P}) & \rightarrow & f_{(t-1,\mathbf{Q})}(\mathbf{P})^{\frac{p^k-1}{\ell}} \end{array} \right.$$

où t est la trace du Frobenius sur $\mathcal{C}(\mathbb{F}_p)$ et est donc de l'ordre de $\sqrt{p}$. On peut noter que le couplage de Ate limite le nombre d'itérations de la boucle de Miller (algorithme 20), mais inverse les 2 groupes $\mathbb{G}_1$ et $\mathbb{G}_2$, ce qui implique que la multiplication scalaire nécessaire dans l'algorithme de Miller s'exécute sur $\mathcal{C}(\mathbb{F}_{p^k})$, et non sur $\mathcal{C}(\mathbb{F}_p)$.

Dans la contribution [118], Vercauteren démontre que pour toute courbe elliptique, il existe un algorithme de couplage utilisant les fonctions de Miller tel que le nombre d'itération de la boucle soit égal à $n = \log_2(p)/\phi(k)$, où ϕ est la fonction d'Euler. Celui-ci dépend de la définition de la courbe.

Un autre élément important rentrant en ligne de compte pour l'implémentation d'un couplage efficace est le facteur suivant :

$$\rho = \frac{\log(p)}{\log(\ell)} \tag{4.2}$$

En effet, ce rapport détermine la taille de la courbe elliptique et du corps fini par rapport au niveau de sécurité. Si celui-ci est trop important, la courbe sera surdimensionnée et risque de nuire aux performances du couplage.

4.2.4 Les courbes Barreto-Naehrig

Comme évoqué dans l'introduction, il n'est pas possible de choisir des courbes au hasard pour faire du couplage. En effet, si la notion de couplage est définie pour toutes les courbes,

k est en général de l'ordre de q, et donc le calcul est impossible. Il existe 2 grandes catégories de courbes avec une valeur de k raisonnable. Les premières sont les courbes supersingulières pour lesquelles $k = 2$, si $p \notin \{2; 3\}$. Malheureusement, k est trop petit pour un usage en cryptographie, ce qui nécessite l'utilisation d'un ρ grand et limite la vitesse du couplage et des exponentiations. Par exemple, pour un niveau de sécurité 128 bits, le NIST [101] conseille une taille de $\mathbb{F}_{p^k}$ de 3000, soit $log_2(p) = 1500$. La taille de ℓ étant 256, cela donne $\rho = 6$.

Ainsi, il est préférable d'utiliser la deuxième catégorie, les courbes ordinaires utilisant un degré de plongement adapté, comme celles construites via la technique de la multiplication complexe (un état de l'art est proposé dans [50]). Au niveau 128 bits, il apparaît que les courbes les mieux adaptées aujourd'hui sont les courbes Barreto-Naehrig [19]. Les courbes BN sont une famille de courbes $y^2 = x^3 + a_6$ définies par un paramètre $x \in \mathbb{Z}$ de la manière suivante :

$$\begin{aligned} p &= 36x^4 + 36x^3 + 24x^2 + 6x + 1 \\ \ell &= 36x^4 + 36x^3 + 18x^2 + 6x + 1 \\ t &= 6x^2 + 1 \\ k &= 12 \end{aligned}$$

Bien sûr, la condition d'existence de ces courbes est que p et ℓ soient premiers. La première chose à noter est que $\rho = 1$ donc $\mathcal{C}(\mathbb{F}_p)[\ell] = \mathcal{C}(\mathbb{F}_p)$. Ensuite, le degré de plongement est idéal pour un niveau de sécurité à 128 bits, puisque si $log_2(p) = 256$ alors $log_2(p^k - 1) = 3072$, ce qui correspond aux recommandations du NIST [101].

Le couplage optimal dans le sens de [118] sur les courbes BN a été proposé par [98]. Soit $r = 6x + 2$, la fonction suivante :

$$e_O \left| \begin{array}{rcl} \mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p) \times \mathcal{C}(\mathbb{F}_p)[\ell] & \to & \overline{\mathbb{F}_p}[\ell] \\ (\mathbf{Q}, \mathbf{P}) & \to & \big((f_{r,\mathbf{Q}}) \cdot g_{(r\mathbf{Q},\pi(\mathbf{Q}))} \cdot g_{(r\mathbf{Q}+\pi(\mathbf{Q}),-\pi^2(\mathbf{Q}))})(\mathbf{P})\big)^{\frac{p^{12}-1}{\ell}} \end{array} \right.$$

définit un couplage sur les courbes BN, appelé couplage Optimal Ate.

Définition 12 *Soit $\mathcal{C}$ une courbe définie sur $\mathbb{F}_p$. Une tordue de $\mathcal{C}$ est une courbe $\mathcal{C}'$ définie sur $\mathbb{F}_p$ telle que $\mathcal{C}$ et $\mathcal{C}'$ soient isomorphes sur $\overline{\mathbb{F}_p}$.*

Soit ζ un élément de $\mathbb{F}_{p^2}$ qui ne soit ni un carré ni un cube dans $\mathbb{F}_{p^2}$. L'image de $\mathcal{C}$ par l'application suivante :

$$\Phi_\zeta \left| \begin{array}{rcl} \mathbb{P}^2(\mathbb{F}_{p^{12}}) & \to & \mathbb{P}^2(\mathbb{F}_{p^{12}}) \\ (x, y, z) & \to & (\frac{x}{\zeta^{1/3}}, \frac{y}{\zeta^{1/2}}) \end{array} \right.$$

est une tordue $\mathcal{C}'$ de $\mathcal{C}$ d'équation $y^2 = x^3 + a_6\zeta$.

L'isomorphisme Φ_ζ est extrêmement intéressant pour les couplages. En effet l'image du groupe $\mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p)$ par Φ_ζ est inclus dans $C'(\mathbb{F}_{p^2})$. On parle alors de tordue de degré 6 puisqu'elle divise la taille de l'extension nécessaire pour représenter les points de $\mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p)$ par 6. Ainsi, la multiplication scalaire sur $\mathcal{C}(\mathbb{F}_{p^k})[\ell] \cap \mathrm{Ker}(\pi - p)$ est remplacée par une multiplication scalaire sur $C'(\mathbb{F}_{p^2})$ beaucoup moins onéreuse.

Ce n'est pas l'unique avantage de la tordue pour l'implémentation du couplage. En effet, comme nous le verrons par la suite, si nous choisissons une représentation de $\mathbb{F}_{p^{12}}$ suivante :

$$\mathbb{F}_{p^{12}} = \frac{\mathbb{F}_{p^2}[X]}{(X^6 - \zeta)\mathbb{F}_{p^2}[X]} \tag{4.3}$$

alors pour tout $\mathbf{P}, \mathbf{Q}, \mathbf{R} \in \big(\mathcal{C}(\mathbb{F}_{p^{12}})[\ell] \cap \mathrm{Ker}(\pi - p)\big)^2 \times \mathcal{C}(\mathbb{F}_p)$ la valeur $g_{(\mathbf{P},\mathbf{Q})}(\mathbf{R})$ est un élément de $\mathbb{F}_{p^{12}}$ dont la moitié des termes sont nuls. Enfin, la verticale $h_{(\mathbf{P})}(\mathbf{R})$ est un

élément de $\mathbb{F}_{p^6}$. Or, comme 12 est le plus petit élément $k \in \mathbb{N}$ tel que ℓ divise $p^k - 1$, $p^6 - 1$ divise $(p^{12} - 1)/\ell$, ce qui implique que $h_{(\mathbf{P})}(\mathbf{R})^{(p^{12}-1)/\ell} = 1$, et donc il n'est pas nécessaire de calculer la fonction h dans l'algorithme de Miller du fait de l'exponentiation finale.

4.2.5 Choix de courbes et extension de corps

La sélection du paramètre x a un impact important sur les performances du couplage. En effet, si sa taille définit le niveau de sécurité atteint, sa structure détermine aussi l'efficacité du calcul du couplage en influant sur 2 aspects :

- le poids de Hamming de $r = 6x + 2$ définit le nombre de fois où la partie additive de la boucle de Miller sera exécutée (branche 6 de l'algorithme 20). Par conséquent celui-ci doit être le plus faible possible. Par ailleurs, le poids de Hamming de x agit aussi sur le temps de calcul dans l'exponentiation finale.
- la valeur de p obtenue doit permettre de construire une extension $\mathbb{F}_{p^2}/\mathbb{F}_p$, et aussi de choisir ζ une extension de $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^2}$, dans laquelle l'arithmétique est aisée.

Dans [103], les considérations pour bien choisir la valeur de x sont caractérisées. Dans ce travail nous avons choisi 3 courbes. Toutes ont le paramètre $a_6 = 2$. La première, nommée BN_{126}, est définie par $x = -(2^{62} + 2^{55} + 1)$ et est déjà apparue dans la littérature publique dans [103, 12]. Elle ne procure qu'une sécurité de 126 bits, mais le poids de Hamming de r n'est que de 5, tandis que $\mathbb{F}_{p^{12}}$ peut être défini comme la tour d'extensions suivante :

$$\begin{array}{lcl} \mathbb{F}_{p^2} & = & \mathbb{F}_p[\mathbf{i}]/(\mathbf{i}^2 + 1) \\ \mathbb{F}_{p^6} & = & \mathbb{F}_{p^2}[\boldsymbol{\beta}]/(\boldsymbol{\beta}^3 - (1 + \mathbf{i})) \\ \mathbb{F}_{p^{12}} & = & \mathbb{F}_{p^6}[\boldsymbol{\Gamma}]/(\boldsymbol{\Gamma}^2 - \boldsymbol{\beta}) = \mathbb{F}_{p^2}[\boldsymbol{\gamma}]/(\boldsymbol{\gamma}^6 - (1 + \mathbf{i})) \\ \zeta & = & 1 + \mathbf{i} \end{array}$$

Ainsi, nous représentons les éléments de $\mathbb{F}_p$ dans la base polynomiale $\mathbf{i}^\alpha \boldsymbol{\gamma}^\beta$ avec $(\alpha, \beta) \in [0; 1] \times [0; 5]$.

Comme le montre l'implémentation logicielle de [12], l'avantage d'utiliser cette courbe est que même quand la réduction fainéante est utilisée, toutes les valeurs à réduire peuvent être maintenues dans 8 registres de 64 bits, sans nécessairement utiliser des registres supplémentaires, qui ralentiraient forcément le calcul. Dans notre cas, les architectures FPGA n'ont pas ce type de contrainte car la taille des multiplieurs peut être étendue à 36 bits. Par conséquent nous avons considéré une courbe BN_{128}, définie par $x = -(2^{63}+2^{22}+2^{18}+2^7+1)$ qui garantit un niveau de sécurité de 128 bits. Enfin, nous proposons la première implémentation matérielle d'un couplage à un niveau de sécurité 192 bits. BN_{192} est définie par $x = -(2^{160} + 2^{74} + 2^{12} + 1)$. Cette courbe présente un niveau de sécurité de 192 bits, toujours selon [101]. Le choix de cette courbe n'est naturellement pas idéal pour le niveau de sécurité, la courbe étant surdimensionnée pour obtenir un corps fini de la bonne taille.

Pour les 3 courbes, l'extension de corps et la base de $\mathbb{F}_{p^{12}}$ sont définies de la même manière. Nous verrons qu'elle permet une arithmétique rapide.

4.3 Calcul de couplage avec un Cox-Rower

Comme nous venons de le voir précédemment, le calcul d'un couplage sur une courbe Barreto-Naehrig consiste essentiellement en un ensemble de calculs dans des extensions de corps finis. Or, l'utilisation de la réduction fainéante permet de réduire de manière importante la complexité d'une multiplication dans une extension, comme constaté dans [12]. Or, comme nous l'avons vue dans le chapitre 3, l'utilisation du RNS permet de réduire

la complexité de la réduction fainéante. Il paraît donc naturel de chercher à utiliser le RNS pour le calcul du couplage.

Dans cette section, nous présentons une adaptation de l'architecture du Cox-Rower utilisée dans le chapitre 3. En effet, l'architecture présentée dans le chapitre 3 ne permet pas telle quelle de réaliser un couplage. La première limitation concerne la structure de la mémoire, la deuxième concerne la taille des bases.

4.3.1 Paramétrisation du Cox-Rower pour le couplage

Comme au chapitre 3, nous définissons 2 bases RNS $\mathcal{B} = (m_1; \cdots; m_n)$ et $\mathcal{B}' = (m'_1; \cdots; m'_n)$ de même taille n, suffisante pour réaliser l'algorithme de couplage. Le dimensionnement de ces bases est à redéfinir, la valeur α définie par l'algorithme 11 devant être revue à la hausse. En effet, durant une multiplication sur $\mathbb{F}_{p^{12}}$, la valeur à réduire peut atteindre $198p^2$, il est donc nécessaire que $\mathcal{M}(\mathcal{B})$ soit supérieur à $198p$ (voir la section 4.4). Les conclusions du chapitre 3 concernant le dimensionnement des multiplieurs restant valide, nous visons $r < 36$. Pour la courbes BN_{126}, n peut donc être ajusté à 6 et r à 33. Pour BN_{128}, r doit être porté à 34. Enfin, BN_{192} nécessite des bases de 19 éléments, avec $r = 35$.

Les éléments de chaque base RNS sont choisis comme des pseudo-Mersenne $m_i = 2^r - \varepsilon_i$, avec $0 \leq \varepsilon_i < 2^{\lfloor r/2 \rfloor}$.

Contrairement aux courbes elliptiques, le nombre de variables locales ne peut pas être maintenu à 8. En effet, comme nous le verrons dans le chapitre 4.4, l'algorithme le plus efficace disponible pour réaliser l'exponentiation finale, qui est donné dans [112] utilise au maximum 8 variables temporaires dans $\mathbb{F}_{p^{12}}$, soit 24 registres par variables (dans $\mathcal{B}$ et $\mathcal{B}'$). Ainsi, il n'est pas possible d'implémenter dans la logique ces mémoires. En revanche, la condition qui nous avait poussé à faire ce choix dans le chapitre 3 n'existe plus dans le couplage. En effet, le parallélisme entre les variables temporaires étant beaucoup plus important, nous pouvons augmenter substantiellement la taille du pipeline, et donc utiliser les mémoires embarquées des FPGA, qui intègrent des buffers en entrée de celles-ci.

Afin de simplifier le séquenceur, nous utilisons des RAM triple port intégrant l'ensemble des précalculs nécessaires aux changements de base (tableau 2.1), ainsi que les variables temporaires des calculs de couplage. La taille de celles-ci est portée à 256. Cette valeur est suffisante pour le support de l'ensemble des courbes proposées dans la section 4.2.5. Les technologies FPGA Altera du nœud technologique 65 nm disposent en effet de mémoires M4k double port de 4608 bits, configurables en 128 mots de 36 bits, tandis que les FPGA supérieurs disposent des RAM M8k double port disposant d'une capacité double des premières. Par conséquent, chaque RAM triple port de chaque Rower consomme 4 M4k (resp. 2 M8k).

4.3.2 Architecture du pipeline

Je propose de redessiner un pipeline pour le Rower permettant d'accélérer [61]. Afin de conserver une vitesse de pipeline équivalente à celle obtenue dans la multiplication scalaire sur courbes elliptiques, l'introduction de matériel supplémentaire doit être réalisée de manière attentive, pour garder le chemin critique sous contrôle. En revanche, il est possible d'augmenter largement la profondeur du pipeline sans risquer de perdre des cycles à cause de l'attente de pipeline. En effet, le couplage présente un parallélisme inhérent bien supérieur à la multiplication scalaire classique sur courbe définie sur $\mathbb{F}_p$. L'architecture de Nozaki [99] ne propose qu'une profondeur de 2 tandis que le chapitre 3 [61] propose l'utilisation d'un pipeline à 5 étages. Pour l'implémentation de couplages, nous pouvons augmenter

la profondeur du pipeline jusqu'à 10 étages et garder le pipeline plein sur l'ensemble du couplage à l'exception de l'inversion sur $\mathbb{F}_p$ (voir section 4.4).

Le tableau suivant indique les différences entre le pipeline du chapitre 3 et celui spécifique au couplage. La figure 4.1 propose une vision schématique du Rower obtenu.

L'état $U(t)$ du pipeline originel caractérisé par l'équation 2.6 est remplacé par un doublet $(U(t); V(t))$ et une sortie $S(t)$, respectant

$$\begin{aligned} U(t)(X_0, X_1, C, \varepsilon, \alpha, \beta, \gamma) &= |9p^2(1-\delta) + \gamma\beta(X_0 \times X_1 + (2-\beta)C) + \alpha U(t-1)|_{2^r-\varepsilon} \\ V(t)(X_0, X_1, C, \varepsilon, \alpha, \beta, \delta) &= |9p^2(1-\delta) + \delta\beta(X_0 \times X_1 + (2-\beta)C) + \alpha V(t-1)|_{2^r-\varepsilon} \\ S(t) &= U(t) \text{ ou } V(t) \end{aligned}$$

Avec $\alpha \in \{0; 1\}$, $\beta \in \{1; 2\}$ et γ et $\delta \in \{-1; 1\}$.

TABLE 4.1 – Description du pipeline et comparaison avec le chapitre 3

	MSCE	Couplage
étage 1	$MSB_1 \| \| LSB_1 = X_0 \times X_1$	$MSB_1 \| \| LSB_1 = X_0 \times X_2$
étage 2	$MSB_2 \| \| LSB_2 = MSB_1 \times \varepsilon$ et $(SR2 = \|LSB1 + C\|_{2^r-\varepsilon})$	$(MSB_2 \| \| LSB_2 = MSB_1 \times \varepsilon$ et $SR2 = \|LSB1 + C\|_{2^r-m})$ ou $(MSB_2 \| \| LSB_2 = MSB_1 \times 2\varepsilon$ et $SR2 = \|2LSB1\|_{2^r-\varepsilon})$
étage 3	$LSB3 = MSB_2 \times \varepsilon$ et $SR3 = \|LSB2 + SR2\|_{2^r-\varepsilon}$	$LSB3 = MSB_2 \times \varepsilon$ et $SR3 = \|LSB2 + SR2\|_{2^r-\varepsilon}$
étage 4	$(acc_1 = \|acc_1 + LSB3\|_{2^r-\varepsilon}$ ou $acc_1 = LSB3)$ et $(acc_2 = \|acc_2 + SR3\|_{2^r-\varepsilon}$ ou $acc_1 = SR3)$	$SR4 = \|SR3 + LSB3\|_{2^r-\varepsilon}$
étage 5	$o = \|acc_1 + acc_2\|_{2^r-\varepsilon}$	$(SR5_1 = \|18p^2 - SR4\|_{2^r-\varepsilon}$ ou $SR5_1 = SR4)$ et $(SR5_2 = \|18p^2 - SR4\|_{2^r-\varepsilon}$ ou $SR5_2 = SR4)$
étage 6		$(acc_1 = \|acc_1 + SR5_1\|_{2^r-\varepsilon}$ ou $acc_1 = SR5_1)$ et $(acc_2 = \|acc_2 - SR5_2\|_{2^r-\varepsilon}$ ou $acc_2 = SR5_2)$
étage 7		$o = acc_1$ ou $o = acc_2$

Les 4 premiers étages calculent $|X_0 \times X_1 + C|_{2^r-\varepsilon}$, mais peuvent aussi réaliser $|2X_0 \times X_1|_{2^r-\varepsilon}$ ce qui permet d'accélérer les carrés dans les extensions de $\mathbb{F}_p$, en calculant des doubles produits en une seule passe. Trois multiplieurs ($r \times r$, $r \times (\mathbf{q}+1)$ et $(\mathbf{q}+1) \times \mathbf{q}$) sont nécessaires, avec $\mathbf{q} = \lceil \log_2 \varepsilon_i \rceil$.

Deux accumulateurs indépendants sont implémentés au $6^{ème}$ étage de pipeline. Ils sont précédés par deux soustracteurs (permettant d'accumuler $|X_0 \times -X_1|_{2^r-\varepsilon}$) et un multiplexeur au $5^{ème}$ étage. Ainsi, la sortie du $4^{ème}$ étage peut être indépendamment additionnée ou soustraite sur les 2 accumulateurs. Par ailleurs chaque accumulateur peut être réinitialisé à chaque cycle. La sortie du pipeline est un simple multiplexeur entre les 2 accumulateurs. Comme nous le verrons dans la section 4.4, ces ajouts permettent une accélération substantielle du couplage suivant l'extension de $\mathbb{F}_p$ définie dans la section 4.2.5.

Sur cette architecture, une multiplication dans $\mathbb{F}_p$ coûte 2 cycles et le résultat peut être accumulé, doublé ou inversé immédiatement. Une réduction coûte par ailleurs $2n+3$ cycles, identiquement à la section 4.

4.4 Arithmétique sur les extensions de $\mathbb{F}_p$

Dans cette section, nous traitons des différentes optimisations utilisées à toutes les étapes du pipeline.

4.4.1 Arithmétique sur $\mathbb{F}_{p^2}$: retour à la méthode naïve

La multiplication de Karatsuba et les différentes techniques d'interpolation développées dans le chapitre 2 ont été intensivement utilisées dans le calcul du couplage pour économiser des multiplications sur $\mathbb{F}_p$, qui dans le cas d'une représentation binaire coûtent un prix équivalent à une réduction. Nous pouvons ainsi citer les contributions de [12, 25, 77, 112]. Karatsuba permet d'économiser 1 multiplication sur 4, mais au prix de 3 additions supplémentaires. En représentation binaire, cette méthode est intéressante sur $\mathbb{F}_{p^2}$, car le coût des 3 additions ne compense pas complètement le gain de la multiplication. En RNS, c'est le contraire, car le coût de la multiplication et de l'addition est équivalent. Ainsi, la méthode naïve de multiplication, dite aussi méthode "schoolbook", limite le nombre d'opérations, et le nombre de cycles sur le Cox-Rower décrit dans la partie 4.3.2 pour une multiplication dans $\mathbb{F}_{p^2}$ est limité à 8 cycles et 2 réductions. Grâce à l'apport de la capacité de doublement du pipeline, un carré dans $\mathbb{F}_{p^2}$ coûte seulement 6 cycles, toujours en utilisant la méthode "schoolbook".

4.4.2 Arithmétique sur $\mathbb{F}_{p^{12}}$, étude de l'apport des techniques d'interpolation

Nous considérons $X = (x_1; \cdots; x_{12})$, $Y = (y_1; \cdots; y_{12})$ et $Z = X \times Y \in \mathbb{F}_{p^{12}}$, dans la représentation donnée par la tour d'extension définie dans la section 4.2.5. Pour accélérer l'arithmétique sur $\mathbb{F}_{p^{12}}$, nous nous donnons comme objectif d'exécuter une seule fois $x_i \times y_j$, pour tout $\{i,j\} \in \{1; \cdots; 12\}^2$. L'extension de $\mathbb{F}_{p^{12}}$ est définie par $\mathbf{i}^2{=}{-}1$ et $\gamma^6{=}(1+\mathbf{i})$, ainsi le résultat $x_i \times y_j$ peut être soit additionné, soit soustrait indépendamment sur 2 coordonnées de Z. C'est la raison principale de la présence des deux accumulateurs indépendants, et du soustracteur associé, qui permettent d'obtenir ce résultat. En outre, le résultat de la multiplication $x_i \times y_j$ peut être multiplié par 2 au moment du passage dans le pipeline avant d'être accumulé (en positif ou en négatif), ce qui permet d'accélérer les doubles produits. Ainsi, le pipeline tel qu'il est défini garantit que le nombre de cycles nécessaires à une multiplication ou un carré sur $\mathbb{F}_{p^{12}}$ est au maximum de 2 fois le nombre de multiplications nécessaires dans l'algorithme "schoolbook".

Nous devons estimer si les méthodes d'interpolation, comme Karatsuba sur $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}$, ou Karatsuba/Toom-Cook sur $\mathbb{F}_{p^6}/\mathbb{F}_{p^2}$ ou $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^4}$, permettent de gagner des cycles par rapport à l'algorithme "schoolbook". La réduction de chacune des coordonnées de $\mathbb{F}_{p^{12}}$ est exclue du comptage de cycles sur cette partie. Par ailleurs, nous ne considérons que Karatsuba défini sur $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}$, qui est le cas le plus favorable pour les techniques d'interpolation. En effet, l'interpolation aux autres niveaux ne peut que donner des résultats moins favorables.

Mais tout d'abord, il est nécessaire de réaliser une taxonomie des différentes multiplications et carrés dans $\mathbb{F}_{p^{12}}$ rencontrés durant le couplage. Il n'en existe en réalité que 4 types :

– Le calcul de carré à chaque étape de la boucle de Miller. Ce carré n'a aucune caractéristique particulière. Grâce à la multiplication par 2 et au double accumulateur intégré dans le pipeline, un carré schoolbook coûte 156 cycles (13×12). La variante de Karatsuba pour un carré permet d'économiser une multiplication dans $\mathbb{F}_{p^6}$ contre 4 additions, suivant l'équation :

$$(X_H + \mathbf{\Gamma} X_L)^2 = \big((X_H + X_L)(X_H + \mathbf{\Gamma}^2 X_L) - (1 + \mathbf{\Gamma}^2) X_H X_L\big) + \mathbf{\Gamma}(2 X_H X_L) \quad \text{avec } X_H, X_L \in \mathbb{F}_{p^6}.$$

Aucune technique n'a été trouvée permettant de réaliser ce calcul dans un temps inférieur aux 156 cycles, sans ajout de matériel spécifique à cette étape.

– Multiplication par la tangente $g_{(\Phi_\zeta(T),\Phi_\zeta(T))}$ dans la boucle de Miller : dans cette multiplication, la moitié des termes de Y est nulle, ce qui avantage fortement la méthode schoolbook. La multiplication coûte ainsi 144 cycles (12×12). Les techniques d'interpolation ne permettent pas d'exploiter la moitié de la taille des opérandes à 0.

– Carrés durant l'exponentiation finale (notamment durant la partie difficile, voir section 4.4.4) : les valeurs à multiplier dans cette partie sont le résultat d'une exponentiation par $(p^6 - 1)(p^2 + 1)$. Elles forment ainsi un sous groupe multiplicatif d'ordre $p^4 - p^2 + 1$, appelé sous groupe cyclotomique $C_{p^{12}}$. Ce sous groupe bénéficie de formules de calcul de carrés simplifiés par rapport au cas général, données par [60] et rappelées dans l'algorithme 21. Seuls 84 cycles sont nécessaires pour ce type de carré. Plus récemment, Karabina [81] a proposé une forme compressée des éléments de $C_{p^{12}}$ qui permet de limiter le nombre de réductions nécessaires à 8 au lieu de 12. Malheureusement, seule la formule de carré existe, et non la formule de multiplication, ce qui nécessite une transformation inverse pour réaliser les multiplications. Cette transformation nécessite des inversions dans $\mathbb{F}_p$, dont le prix est prohibitif sur le Cox-Rower. Ces formules ne peuvent pas être utilisées.

Algorithme 22 : Carré durant l'exponentiation finale [60]

Entrée(s): $A = \sum_{i=0}^{5} a_i \boldsymbol{\gamma}^i \in C_{p^{12}}$ avec $a_i \in \mathbb{F}_{p^2}, 0 \leq i \leq 5$
Sortie(s): A^2

1: $A_0 \leftarrow 3a_0^2 + 3(1 + \mathbf{i})a_3^2 - 2a_0, A_1 \leftarrow 6(1 + \mathbf{i})a_2 a_5 + 2a_1$
2: $A_2 \leftarrow 3a_1^2 + 3(1 + \mathbf{i})a_4^2 - 2a_2, A_3 \leftarrow 6a_0 a_3 + 2a_3$
3: $A_4 \leftarrow 3a_2^2 + 3(1 + \mathbf{i})a_5^2 - 2a_4, A_5 \leftarrow 6a_1 a_4 + 2a_5$
4: **retourne** $\sum_{i=0}^{5} A_i \boldsymbol{\gamma}^i$

– Multiplication durant l'exponentiation finale : ici, aucune structure des opérandes ne permet de trouver des formules simplifiées. Sans compter les attentes nécessaires au pipeline à cause de la multiplication des résultats intermédiaires, Karatsuba nécessite 302 cycles sur le circuit proposé dans ce chapitre tandis que la méthode "schoolbook" calcule le même résultat en 288 cycles. Ce résultat peut être descendu à 278 si une capacité d'addition en 1 cycle est donnée au Rower (par exemple, par l'ajout d'un multiplexeur permettant de positionner C à une valeur de ROM_2 ou de la mémoire principale), voire même à 254 cycles si l'addition peut s'effectuer en même temps que la multiplication (via l'ajout de mémoires spécifiques). Néanmoins, seules 20 opérations de ce type sont nécessaires sur l'ensemble du couplage pour BN_{126} et 26 pour BN_{128}, soit un gain modéré de 200 à 680 cycles pour la première et 260 à 884 cycles pour la deuxième sur l'ensemble du couplage, sans compter les attentes de pipeline et l'ajout de matériel. Au vu du gain potentiel limité, nous avons décidé de ne pas implémenter cette technique.

4.4.3 Inversion sur $\mathbb{F}_{p^{12}}$

Le calcul du couplage nécessite une inversion dans $\mathbb{F}_{p^{12}}$ située dans exponentiation finale (voir section 4.4.4). En réalité, il ne s'agit pas d'une inversion mais d'un calcul de f^{p^6-1}, ce qui est équivalent en terme de complexité. L'algorithme 23 donne le séquencement du calcul de f^{p^6-1} pour $f \in \mathbb{F}_{p^{12}}$. Cet algorithme, par calcul successif de conjugués dans la tour d'extension de $\mathbb{F}_{p^{12}}$, remplace l'inversion dans $\mathbb{F}_{p^{12}}$ par une unique inversion dans $\mathbb{F}_p$, accompagnée de réductions et multiplications supplémentaires [43, 77].

Algorithme 23 : Calcul de f^{p^6-1}

1: $\tilde{f} = \text{conjugué}_{12}(f)$ $A = f.\tilde{f}(\in \mathbb{F}_{p^6})$
2: $\tilde{A} = \text{conjugué}_6(A)$ $B = A\tilde{A}(\in \mathbb{F}_{p^2})$
3: $\tilde{C} = \text{conjugué}_2(B)$ $D = C\tilde{C}(\in \mathbb{F}_p)$
4: **retourne** $D^{-1}\tilde{C}\tilde{B}\tilde{A}\tilde{f}^2$

L'inversion restante dans $\mathbb{F}_p$ reste néanmoins extrêmement chère, puisqu'elle est exécutée dans un Cox-Rower. En effet, la comparaison étant chère en RNS, l'inversion ne peut pas être réalisée par le classique algorithme d'Euclide étendu, mais par une exponentiation onéreuse par $p-2$. Pour cette opération, l'algorithme "square et multiply" est utilisé en faisant attention de choisir la méthode Least Significant Bit donnée par l'algorithme 8. En effet, même s'il consomme plus de mémoire, cette variante permet de paralléliser les multiplications dans le cas où le bit de l'exposant est à 1, car il est possible de glisser la multiplication et la réduction dans les attentes du pipeline dues à la réduction du carré.

4.4.4 Séquencement de plus haut niveau

L'algorithme 24 définit le couplage "Twisted Optimal Ate" sur $\mathcal{C}$ qui est la fonction définie par

$$\hat{e}_O \left| \begin{array}{l} \mathcal{C}'(\mathbb{F}_{p^2})[\ell] \cap \text{Ker}(\pi - p) \times \mathcal{C}(\mathbb{F}_p)[\ell] \to \mathbb{F}_{p^{12}}[\ell] \\ (\mathbf{Q}, \mathbf{P}) \to \Big((f_{r,\Phi_\zeta(\mathbf{Q})} \cdot g_{(\Phi_\zeta(r\mathbf{Q}),\Phi_\zeta(\pi(\mathbf{Q})))} \cdot g_{(\Phi_\zeta(r\mathbf{Q})+\Phi_\zeta(\pi(\mathbf{Q})),\Phi_\zeta(-\pi^2(\mathbf{Q})))})(\mathbf{P}) \Big)^{\frac{p^{12}-1}{\ell}} \end{array} \right.$$

Ainsi, un élément de $\mathbb{G}_1$ est représenté par les coordonnées $(x_\mathbf{Q}; y_\mathbf{Q}) \in \left(\mathbb{F}_{p^2}\right)^2$, et $\mathbb{G}_2$ par les coordonnées $(x_\mathbf{P}; y_\mathbf{P}) \in \mathbb{F}_p^2$. Le résultat est un élément de $\mathbb{F}_{p^{12}}$.

En général, les coordonnées Jacobiennes sont utilisées dans les couplages car elles permettent de limiter le nombre d'opérations chères dans $\mathbb{F}_{p^2}$ comme les multiplications et les carrés [24, 77, 98]. Néanmoins, dans notre cas, il est plus judicieux d'utiliser les coordonnées projectives quand la réduction fainéante est utilisée, comme l'indiquent [12, 40]. L'avantage que procurent les coordonnées projectives par rapport aux jacobiennes est amplifié quand le RNS est utilisé. Les algorithmes 25 et 26 permettent de calculer sur l'ensemble des courbes choisies dans la section précédente les sous-fonctions `add` et `dbl` de l'algorithme 24. Les opérations sont ici réarrangées pour mettre en valeur les réductions et le parallélisme de ces algorithmes. En effet, chaque ligne de ces 2 algorithmes peut être calculée dans un ordre aléatoire, car il n'existe aucune dépendance entre les résultats intermédiaires.

L'algorithme `dbl` calcule non seulement le doublement du point $\mathbf{T}$, mais aussi l'évaluation au point $\mathbf{P}$ de la tangente en $\Phi_\zeta^{-1}(\mathbf{T})$ de la courbe $\mathcal{C}$. Le coût total de cette étape est de 20 réductions dans $\mathbb{F}_p$ et de 4 multiplications, 5 carrés dans $\mathbb{F}_{p^2}$, ainsi que 2 multiplications d'un élément de $\mathbb{F}_{p^2}$ par un élément de $\mathbb{F}_p$. De même, l'algorithme `add` réalise le calcul de

Algorithme 24 : Couplage Optimal Ate pour les courbes BN pour $x < 0$

Entrée(s):
$\mathbf{P} \in \mathcal{C}(\mathbb{F}_p)[\ell], \mathbf{Q} = (x_\mathbf{Q}, y_\mathbf{Q}) \in \mathcal{C}'(\mathbb{F}_{p^2})[\ell] \cap \mathrm{Ker}(\pi - p), r = ||6x+2|| = \sum_{i=0}^{s-1} r_i 2^i$, et $x < 0$.

Sortie(s): $\hat{e}_O(\mathbf{Q}, \mathbf{P}) \in \mathbb{F}_{p^{12}}$

1: $\mathbf{T} \leftarrow \mathbf{Q}, f \leftarrow 1$
2: **pour** $i = s-2$ downto 0 **faire**
3: $\quad \mathbf{T}, g \leftarrow \mathtt{dbl}(\mathbf{T}, \mathbf{P}), f \leftarrow f^2 \cdot g$
4: $\quad$ **si** $r_i = 1$ **alors**
5: $\quad\quad \mathbf{T}, g \leftarrow \mathtt{add}(\mathbf{T}, \mathbf{Q}, \mathbf{P}), f \leftarrow f \cdot g$
6: $\quad$ **fin si**
7: **fin pour**
8: $\mathbf{T} \leftarrow -\mathbf{T}, f \leftarrow f^{p^6}$ (cette ligne est nécessaire car $x < 0$)
9: $\mathbf{Q}_1 \leftarrow \pi(\mathbf{Q}), \mathbf{Q}_2 \leftarrow -\pi(\mathbf{Q}_1)$
10: $\mathbf{T}, g \leftarrow \mathtt{add}(\mathbf{T}, \mathbf{Q}_1, \mathbf{P}), f \leftarrow f \cdot g$
11: $\mathbf{T}, g \leftarrow \mathtt{add}(\mathbf{T}, \mathbf{Q}_2, \mathbf{P}), f \leftarrow f \cdot g$
12: $f \leftarrow \left(f^{p^6-1}\right)^{p^2+1}$
13: $f \leftarrow \mathtt{hard\text{-}part}(f, |x|)$
14: **retourne** f

$\mathbf{T} + \mathbf{Q}$ et l'évaluation en $\mathbf{P}$ de la droite passant par $\Phi_\zeta^{-1}(\mathbf{T}), \Phi_\zeta^{-1}(\mathbf{Q})$. Son coût total est de 20 réductions dans $\mathbb{F}_p$, 11 multiplications, 2 carrés dans $\mathbb{F}_{p^2}$ et 2 multiplications d'un élément de $\mathbb{F}_{p^2}$ par un élément de $\mathbb{F}_p$.

Algorithme 25 : `dbl`, doublement

Entrée(s): $\mathbf{T} = (X_\mathbf{T}, Y_\mathbf{T}, Z_\mathbf{T}) \in \mathcal{C}'(\mathbb{F}_{p^2})$, $\mathbf{P} = (x_\mathbf{P}, y_\mathbf{P}) \in C(\mathbb{F}_p)$

Sortie(s): Le point2$\mathbf{T}$ et l'évaluation en $\mathbf{P}$ de l'équation de la tangente à $\mathcal{C}$ en $\Phi_\zeta^{-1}(\mathbf{T})$.

1: $B \leftarrow Y_\mathbf{T}^2, C \leftarrow 3(1+\mathbf{i})Z_\mathbf{T}^2, D \leftarrow 2X_T Y_\mathbf{T}$
2: $F \leftarrow B + 3\mathbf{i}C, G \leftarrow B - 3\mathbf{i}C, H \leftarrow 3C, t_3 \leftarrow B + \mathbf{i}C, A \leftarrow X_\mathbf{T}^2, E \leftarrow 2Y_T Z_Y$
3: $X_{2\mathbf{T}} \leftarrow DF, Y_{2\mathbf{T}} \leftarrow G^2 + 4HC, Z_{2\mathbf{T}} \leftarrow 4BE, t_0 \leftarrow Ey_\mathbf{P}, t_1 \leftarrow -3Ax_\mathbf{P}$
4: **retourne** $(X_{2\mathbf{T}}\boldsymbol{\gamma}^2, Y_{2\mathbf{T}}\boldsymbol{\gamma}^3, Z_{2\mathbf{T}}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

Algorithme 26 : `add`, addition

Entrée(s): $\mathbf{T} = (X_\mathbf{T}, Y_\mathbf{T}, Z_\mathbf{T}) \in \mathcal{C}'(\mathbb{F}_{p^2})$, $\mathbf{Q} = (x_\mathbf{Q}, y_\mathbf{Q}) \in \mathcal{C}'(\mathbb{F}_{p^2})$,
$\mathbf{P} = (x_\mathbf{P}, y_\mathbf{P}) \in \mathcal{C}(\mathbb{F}_p)$.

Sortie(s): Le point $\mathbf{T} + \mathbf{Q}$ et l'évaluation en $\mathbf{P}$ de l'équation de la ligne $\Phi_\zeta^{-1}(\mathbf{T})$ et $\Phi_\zeta^{-1}(\mathbf{Q})$..

1: $E \leftarrow x_Q Z_\mathbf{T} - X_\mathbf{T}, F \leftarrow y_Q Z_\mathbf{T} - Y_\mathbf{T}$
2: $E_2 \leftarrow E^2, F_2 \leftarrow F^2$
3: $A \leftarrow F_2 Z_\mathbf{T} - 2X_T E_2 - EE_2, B \leftarrow X_T E_2, E_3 \leftarrow EE_2$
4: $X_{\mathbf{T}+\mathbf{Q}} \leftarrow AE, Z_{\mathbf{T}+\mathbf{Q}} \leftarrow Z_T E_3, t_3 \leftarrow Fx_\mathbf{Q} - Ey_\mathbf{Q}$
5: $Y_{\mathbf{T}+\mathbf{Q}} \leftarrow F(B-A) - y_Q E_3, t_0 \leftarrow Ey_\mathbf{P}, t_1 \leftarrow -Fx_\mathbf{P}$
6: **retourne** $(X_{\mathbf{T}+\mathbf{Q}}\boldsymbol{\gamma}^2, Y_{\mathbf{T}+\mathbf{Q}}\boldsymbol{\gamma}^3, Z_{\mathbf{T}+\mathbf{Q}}), t_0 + t_1\boldsymbol{\gamma} + t_3\boldsymbol{\gamma}^3$

Le calcul de $f^{(p^{12}-1)/\ell}$ est connu dans la littérature sous le nom d'exponentiation finale.

Dans [85], Koblitz and Menezes montrent qu'elle peut être séparée en 2 parties grâce à la factorisation de l'exposant :

$$\frac{p^{12}-1}{\ell} = \left(p^6-1\right)\left(p^2+1\right)\left(\frac{p^4-p^2+1}{\ell}\right) \tag{4.4}$$

Ainsi la partie dite "facile" consiste à calculer $f^{(p^6-1)(p^2+1)}$, ce qui coûte une inversion dans $\mathbb{F}_{p^{12}}$ ainsi que 2 multiplications et un Frobenius carré. La seconde étape de l'exponentiation, dite "difficile", est l'exponentiation d'un élément de $\mathbb{F}_{p^{12}}$ par $h = (p^4-p^2+1)/\ell$. Scott [112] a proposé une factorisation de h permettant une exponentiation intéressante, en optimisant l'utilisation des différents Frobenius. Cette optimisation s'appuie sur la forme spécifique de p et ℓ pour les courbes Barreto-Naehrig.

$$f^h = [f^p f^{p^2} f^{p^3}] \cdot [f^{-2}] \cdot [f^{x^2 p^2}]^6 \cdot [f^{-xp}]^{12} \cdot [f^{-x} \cdot f^{-x^2-p}]^{18} \cdot [f^{-x^2}]^{30} \cdot [f^{-x^3 p+1}]^{36} \tag{4.5}$$

On peut noter l'existence dans l'algorithme 27 de nombreuses inversions. Néanmoins, on peut se rappeler que la valeur d'entrée f de cet algorithme est telle que $\exists f' \in \mathbb{F}_{p^{12}} f = f'^{(p^6-1)(p^2+1)}$ et donc f appartient au sous groupe cyclotomique $C_{p^{12}}$. Les inversions correspondent donc à des calculs de conjugués dans $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}$.

Algorithme 27 : `hard-part`, partie difficile de l'exponentiation finale proposée par [112]

Entrée(s): $f \in C_{p^{12}}$, $u = ||x||$.
Sortie(s): $f^{(p^4-p^2+1)/\ell}$

1: $y_0 \leftarrow f^p f^{p^2} f^{p^3}$, $y_1 \leftarrow f^u$, $y_3 \leftarrow y_1^u$, $y_5 \leftarrow y_3^u$, $y_4 \leftarrow y_5^p$, $y_6 \leftarrow y_4 y_5 \left(= f^{u^3}\left(f^{u^3}\right)^p\right)$
2: $y_5 \leftarrow y_3^p$, $y_2 \leftarrow y_5^{-1}$, $y_4 \leftarrow y_1 y_2 \left(= f^u / \left(f^{u^2}\right)^p\right)$
3: $y_2 \leftarrow y_5^p \left(= \left(f^{u^2}\right)^{p^2}\right)$, $y_5 \leftarrow y_3^{-1} \left(= 1/f^{u^2}\right)$, $y_3 \leftarrow y_1^p\,((m^u)^p)$, $y_1 \leftarrow f^{-1}$
{Chaîne d'addition permettant le calcul de $y_0.y_1^2.y_2^6.y_3^{12}.y_4^{18}.y_5^{30}.y_6^{36}$}
4: $t_0 \leftarrow y_6^2$, $t_0 \leftarrow t_0 y_4$, $t_0 \leftarrow t_0 y_5$, $t_1 \leftarrow y_3 y_5$, $t_1 \leftarrow t_1 t_0$, $t_0 \leftarrow t_0 y_2$, $t_1 \leftarrow t_1^2$
5: $t_1 \leftarrow t_1 t_0$, $t_1 \leftarrow t_1^2$, $t_0 \leftarrow t_1 y_1$, $t_1 \leftarrow t_1 y_0$, $t_0 \leftarrow t_0^2$, $t_0 \leftarrow t_0 t_1$
6: **retourne** t_0

4.4.5 Contrôle des variables locales

Les choix algorithmiques décrits dans cette section décortiquent les opérations présentes dans le calcul du couplage en opérations simples sur $\mathbb{F}_p$. Comme pour le calcul de la multiplication scalaire sur courbes elliptiques, deux contraintes sont à prendre en compte :

- le contrôle du nombre de variables locales présentes en même temps dans l'algorithme. Le point le plus critique se trouve dans l'algorithme `hard-part` de l'exponentiation finale. Ainsi l'algorithme 27 est réarrangé de telle manière qu'il limite le nombre de variables locales à 8, ce qui représente tout de même 192 registres.
- le contrôle de la dépendance entre les opérations, pour limiter les attentes de pipeline. Les contraintes sur ce point sont beaucoup moins importantes que dans le cas de la multiplication scalaire sur courbes elliptiques. Les algorithmes 25 et 26 sont les plus contraints sur ce point, et on peut remarquer qu'il est très facile de réaliser 100% d'occupation de pipeline même dans la boucle de Miller. Seule l'inversion dans $\mathbb{F}_p$

fait exception, où le taux d'occupation du pipeline tombe à environ 40% pour $n = 8$. Les chiffres présentés dans la section 4.5 présentent ainsi un taux d'occupation du pipeline supérieur à 99% pour toutes les courbes en dehors de cette inversion, les quelques cycles non utilisés servant au flot de contrôle de l'algorithme.

4.5 Résultats d'implémentation, analyse et comparaison

4.5.1 Taille du circuit

Des prototypes du coprocesseur décrit dans la section 4.3 et 4.4 ont été implémentés dans des FPGA commerciaux de plusieurs nœuds technologiques. Le Cox-Rower $n = 8$ a été implémenté sur 3 technologies :

- Le EP2C35 : un FPGA Altera "low cost" développé sur le nœud technologique 65 nm FPGA. Son coût à l'unité est inférieur à $100 sur le site [1]. Ce type de composant peut aisément être intégré dans une production de grande série.
- EP2S30 : Un FPGA "high end" du nœud technologique 65 nm d'Altera. C'est celui qui a été utilisé pour le coprocesseur du chapitre 3. Il est utilisé pour une comparaison avec l'implémentation de [45] sur une technologie comparable (le Virtex IV).
- EP3SE50 : Un FPGA haute densité d'Altera du nœud 40 nm. Il s'agit du plus petit FPGA de la série Stratix III. Le Cox-Rower avec 19 Rowers nécessaire à la courbe BN_{192} est aussi implémenté dans ce composant.

Le code source utilisé est le même pour chaque circuit à l'exception de la génération des mémoires spécifiques à chaque technologie, et est généré automatiquement. Le séquenceur est microcodé et le firmware est constitué de 20 ko de microcode, incluant le flot de contrôle et le séquencement au cycle près des Rower. Ce firmware utilise en général les grandes mémoires disponibles dans les FPGA. Le même Cox-Rower $n = 8$ supporte les 2 courbes BN_{126} et BN_{128}, la seule différence entre les 2 coprocesseurs étant les valeurs précalculées se trouvant dans les mémoires de chaque Rower et du séquenceur.

TABLE 4.2 – Consommation des ressources des FPGA utilisés

	n	**Composant**	**Freq.**	**DSP**	**Logique**	**Mémoire**	**Séquenceur**
Circuit générique	8	EP2C35	91 MHz	35 18 mult.	14274 LE	32 M4k	35 M4k
	8	EP2S30	165 MHz	72 DSP18el	4227 ALMs	32 M4k	1 M512
	8	EP3SE50	165 MHz	72 DSP18el	4233 ALMs	16 M9k	1 M144 +2 M9k
	19	EP3SE50	131 MHz	171 DSP18el	9910 ALMs	38 M9k	1 M144 +2 M9k
Circuit optimisé [3]	8	XC6VLX240	250 MHz	32 DSP48E1s	7032 Slices	-	45 18Kb BRAMs

4.5.2 Latence d'un couplage

Le tableau 4.3 rend compte du nombre de cycles utilisés dans chacune des sous fonctions du couplage Optimal Ate sur le circuit présenté dans la section 4.3.

On peut constater que par rapport aux données antérieures de l'état de l'art données par les travaux de Fan et al [47, 46], le RNS permet de gagner des cycles sur chaque opération élémentaire d'un couplage. A la fin, notre circuit utilise 3 fois moins de cycles et le circuit optimisé, présenté en annexe, presque 4. A fréquence égale, le RNS possède un avantage comparatif indéniable pour le calcul du couplage par rapport à la représentation binaire.

3. Le circuit optimisé a été développé en parallèle de ce travail par Fan et al. [121], et fait l'objet d'une

TABLE 4.3 – Coût des opérations intermédiaires dans le couplage optimal Ate

	Courbe sur $\mathbb{F}_p$	Mul./Red. $g_{(\mathbf{T},\mathbf{T})}(\mathbf{P})$	$2\mathbf{T}$ et $g_{(\mathbf{T},\mathbf{Q})}(\mathbf{P})$	$\mathbf{T}+\mathbf{Q}$ et	f^2	$f \cdot g$ Miller	Boucle Exp.	Expo finale	Total
Ce circuit	BN_{126}	2 / 19	507	581	384	372	86,530	89,581	176,111
	BN_{128}						92,480	94,101	192,502
	BN_{192}	2/ 41	947	1153	648	636	401,565	388,284	789,849
Circuit optimisé	BN_{126}	2/ 12	320	430	301	289	61,116	81,995	143,111
Fan et al [47, 46]	autre	-	996	1260	1541	1239	311,418	281,558	592,976

4.5.3 Comparaisons et discussions

Le tableau 4.4 liste les performances logicielles et matérielles publiées dans la littérature scientifique ces dernières années. L'ensemble des implémentations hardwares [46, 54, 80] est surpassée à ce niveau de sécurité par notre implémentation générique RNS à l'exception de [55], dont la consommation en slice est très importante. L'implémentation optimisée permet encore d'augmenter la vitesse, mais ne permet pas une implémentation sur tous les types de FPGA. Bien sûr, les architectures proposées s'implémentent sur des plate-formes différentes et consomment différemment les ressources disponibles. Par conséquent, une comparaison objective est assez difficile.

Les implémentations logicielles restent aujourd'hui plus performantes d'un facteur 2 du circuit générique sur Stratix III, et d'un facteur 1,1 du circuit optimisé. La raison essentielle est le cadencement du multiplieur 64 bits des processeurs x86 modernes. C'est néanmoins la première implémentation sur FPGA à avoir des performances comparables au logiciel pour une primitive cryptographique sur corps de grande caractéristique.

La performance constatée dans ce tableau des implémentations RNS en comparaison des autres types de représentations repose sur 2 avantages complémentaires :

- la complexité du couplage en représentation RNS est très nettement inférieure à la complexité multiprécision, grâce au fait que la majorité de la complexité est reportée sur la réduction, et la multiplication reste linéaire. De ce fait la technique de réduction fainéante est beaucoup plus agressives en RNS qu'en représentation binaire.
- le RNS présente une grande disposition au parallélisme. En effet, l'absence de propagation de retenues entre les canaux des bases permet d'accélérer les circuits sans nécessiter d'ajout de pipeline artificiel. L'architecture Cox-Rower présentée dans le chapitre 3 répond ainsi parfaitement aux besoins de parallélisme et de vitesse.

Ce travail permet de confirmer par une démarche expérimentale les conclusions dessinées par [43] quant à la compétitivité du RNS quand il s'agit de couplage haute performance.

4.6 Conclusion

Ce travail nous a permis de démontrer que le RNS est une approche très compétitive pour l'implémentation efficace de couplages à des niveaux de sécurité importants (supérieurs à 128 bits). Grâce au RNS, l'ensemble des calculs peut être hautement parallélisé, dans une architecture Cox-Rower proche de celle originellement proposée par [71]. Le circuit générique permet de calculer un couplage à un niveau de sécurité de 128 bits, sur un FPGA faible coût en 2 ms, ce qui représente un gain de plus de 2 par rapport à [45]. Les

publication conjointe [35]. Les optimisations sont spécifiques au Virtex 6 et sont présentées en annexe de ce chapitre.

TABLE 4.4 – État de l'art des implémentations orientées vitesse des couplages

contribution	**Couplage**	**Securité** [bit]	**Plate-forme**	**Algorithme**	**taille**	**Freq.** [MHz]	**Cycles** [$\times 10^3$]	**latence** [ms]
Circuit gén.	optimal ate	126	Altera FPGA (Cyclone II)	RNS Montgomery	14274 LE 35 mult.	91	176	1.93
			Altera FPGA (Stratix III)		4233 A 72 DSPs	165	176	1.07
		192	Altera FPGA (Stratix III)		9910 A 171 DSPs	131	790	6.03
Circuit opt.	optimal ate	126	Xilinx FPGA (Virtex-6)	RNS Montgomery	7032 slices 32 DSPs	250	143	0.573
[46]	ate	128	Xilinx FPGA (Virtex-6)	Hybrid Montgomery	4014 slices 42 DSPs	210	336	1.60
	optimal ate						245	1.17
[54]	Tate	128	Xilinx FPGA (Virtex-4)	Blakley	52k Slices	50	1,730	34.6
	ate						1,207	24.2
	optimal ate						821	16.4
[80]	Tate	128	ASIC (130 nm)	Montgomery	97 kGates	338	11,627*	34.4
	ate						7,706*	22.8
	optimal ate						5,340*	15.8
[45]	Tate over $\mathbb{F}_{3^{5\cdot 97}}$	128	Xilinx FPGA (Virtex-4)	-	4755 Slices 7 BRAMs	192	429	2.23
[10]	optimal Eta over $\mathbb{F}_{2^{367}}$	128	Xilinx Virtex-4	-	4518 Slices	220	774*	3.52
[55]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xilinx Virtex-6	-	15167 Slices	250	49	190
[77]	ate	128	64-bit Core2	Montgomery	-	2400	15,000	6.25
	optimal ate						10,000	4.17
[59]	ate	128	64-bit Core2	Montgomery		2400	14,429	6.01
[98]	optimal ate	128	Core2 Quad	Hybrid Mult.	-	2394	4,470	1.86
[25]	optimal ate	126	Core i7	Montgomery	-	2800	2,330	0.83
[12]	optimal ate	126	Phenom II	Montgomery	-	3000	1,562	0.52
[11]	η_T over $\mathbb{F}_{2^{1223}}$	128	Xeon	-	-	2000	3,020	1.51
[23]	η_T over $\mathbb{F}_{3^{509}}$	128	Core i7	-	-	2900	5,423	1.87
[10]	opt. Eta $\mathbb{F}_{2^{367}}$	128	Core i5	-	-	2530	2,440	0.96

* Estimation de [35]

implémentations sur les FPGA haute densité permettent d'avoir une implémentation aisément portable sur n'importe quel FPGA dont la vitesse excède la meilleure optimisation proposée jusqu'alors dans la littérature [46], optimisée pour un Virtex 6. Une implémentation optimisée sur un Virtex 6 telle que celle qui est proposée en annexe permet d'obtenir un facteur 2 supplémentaire.

Enfin, nous proposons la première implémentation matérielle du couplage à 192 bits de sécurité, en utilisant les mêmes courbes Barreto-Naehrig. 6 ms suffisent à implémenter un couplage sur cette courbe avec le circuit proposé dans ces travaux. Cette implémentation doit servir de point de repère pour les travaux futurs, notamment les implémentations logicielles, ainsi que la recherche de courbes optimales à ce niveau de sécurité, question qui reste encore ouverte à ce jour.

4.7 Annexe A : circuit optimisé pour la courbe BN_{126} sur FPGA Virtex 6

Cette annexe traite du circuit appelé circuit optimisé publié dans ma contribution [61] dont l'idée originale a été apportée par mes coauteurs, à l'exception de Sylvain Duquesne et moi-même. Il ne s'agit pas à proprement parlé de ma contribution directe, celle-ci étant le circuit dit "générique". Néanmoins, de nombreux échanges avec mes coauteurs ont permis d'améliorer leur résultat de 18% par rapport à leur contribution originale [121]. Enfin, par souci de complétude sur le sujet des implémentations hardware utilisant le RNS pour le couplage, je présente dans cette section les idées essentielles permettant d'arriver à ce résultat.

La première idée permettant de gagner 19% du nombre total de cycles est l'utilisation d'une base RNS particulière. Celle-ci est donnée par $r = 33$ et

$$\begin{array}{lllllllll} \mathcal{B} & = (2^r-1, & 2^r-9, & 2^r+3, & 2^r+11, & 2^r+5, & 2^r+9, & 2^r-31, & 2^r+15), \\ \mathcal{B}' & = (2^r, & 2^r+1, & 2^r-3, & 2^r+17, & 2^r-13, & 2^r-21, & 2^r-25, & 2^r-33). \end{array}$$

On peut constater que cette base répond aux exigences de choix de base développées dans le chapitre 3 pour BN_{126}. Néanmoins, l'utilisation de nombres supérieurs à 2^r nécessite une petite adaptation par rapport à l'algorithme 11 : lors du premier changement de base, *errinit* ne doit pas être positionnée à 0, mais à une valeur négative afin de s'assurer que p n'est pas retiré une fois de trop sur le résultat final S(qui se retrouverait être négatif).

L'avantage de l'utilisation d'une telle base réside dans les précalculs qui lui sont associés. En effet, si nous appelons taille en bits utiles de r la valeur suivante $b(r) = \lceil log_2(abs(r >> (max(d \text{ tq } 2^d | r)))) \rceil$, soit la longueur entre le premier bit non nul d'un nombre signé et le dernier, alors la matrice $M_{i,j} = b(|\mathcal{M}(\mathcal{B})/m_i|_{m'_j}))$ et $\tilde{M}_{i,j} = b(|\mathcal{M}(\mathcal{B})/m_i|_{m'_j})$, nous obtenons :

$$M = \begin{pmatrix} 23 & 20 & 21 & 20 & 21 & 20 & 18 & 19 \\ 12 & 10 & 12 & 10 & 11 & 10 & 8 & 9 \\ 16 & 14 & 14 & 13 & 14 & 13 & 12 & 13 \\ 15 & 14 & 15 & 16 & 15 & 16 & 13 & 18 \\ 17 & 18 & 16 & 16 & 16 & 16 & 16 & 15 \\ 20 & 21 & 20 & 19 & 19 & 19 & 21 & 19 \\ 19 & 20 & 19 & 19 & 19 & 19 & 21 & 19 \\ 19 & 19 & 19 & 18 & 19 & 19 & 23 & 18 \end{pmatrix}, M' = \begin{pmatrix} 14 & 13 & 13 & 10 & 11 & 10 & 10 & 9 \\ 15 & 15 & 16 & 14 & 16 & 15 & 14 & 14 \\ 16 & 17 & 15 & 14 & 14 & 13 & 13 & 13 \\ 20 & 21 & 20 & 21 & 19 & 19 & 19 & 18 \\ 20 & 20 & 19 & 19 & 18 & 18 & 18 & 17 \\ 21 & 21 & 21 & 21 & 20 & 19 & 19 & 19 \\ 17 & 17 & 17 & 16 & 18 & 19 & 19 & 21 \\ 20 & 20 & 20 & 23 & 19 & 19 & 19 & 19 \end{pmatrix}$$

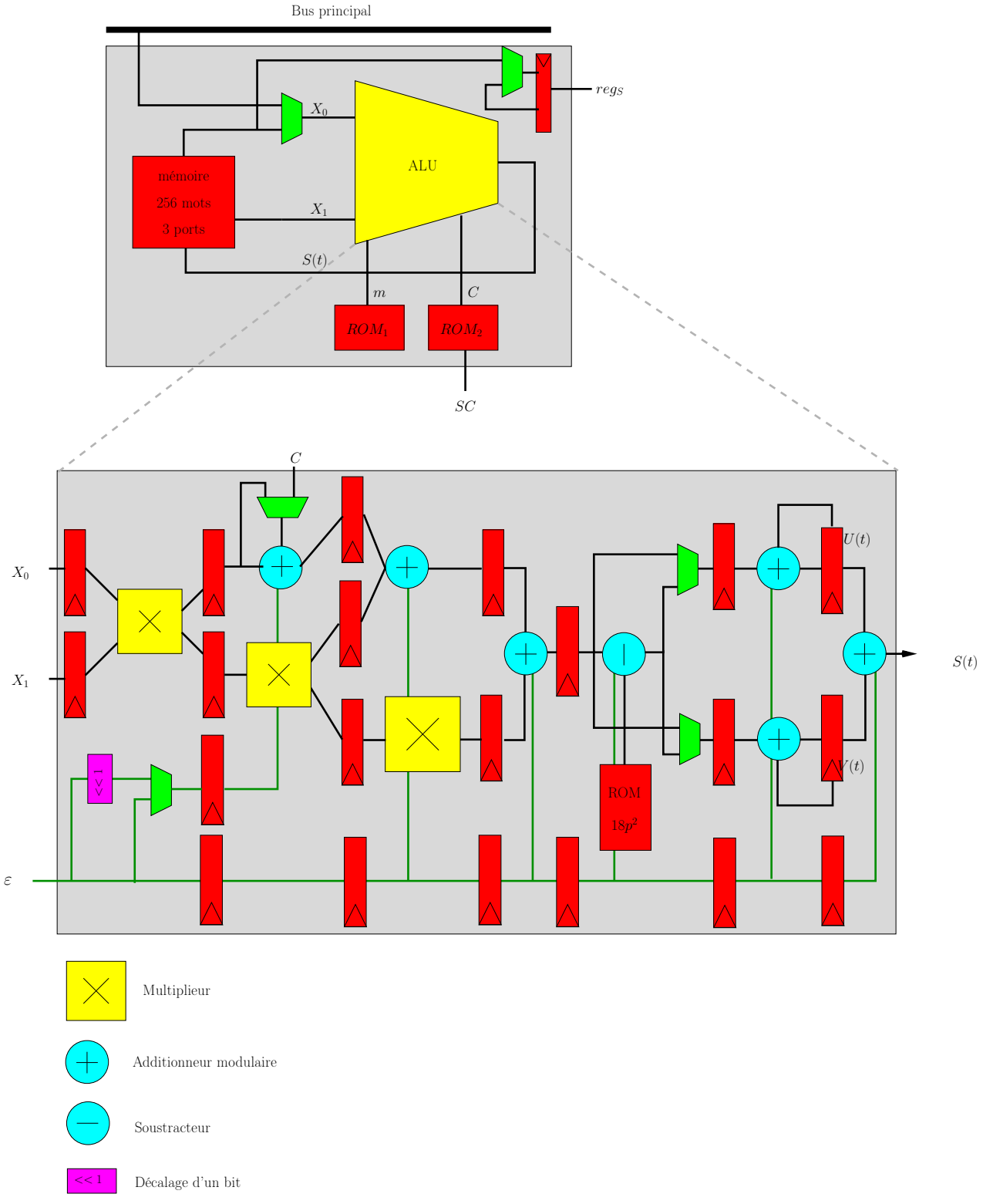
Ainsi, l'ensemble des précalculs utilisés dans les changements de base ont une taille inférieure à 25 bits, ce qui permet d'utiliser le bloc DSP 36×42 du Virtex 6 en 2 multiplieurs signés indépendants 36×25 dans le même cycle. L'avantage d'une telle approche est qu'elle limite le nombre d'itérations de la boucle du changement de base à 4 au lieu de 8. En revanche, il n'est plus possible de mixer les précalculs comme cela était réalisé dans l'architecture générique. Par conséquent, une réduction modulaire est réalisée en 12 cycles au lieu de 19.

La structure des bases permet par ailleurs d'économiser les blocs DSP sur le bloc de réduction modulaire, ces éléments ayant seulement un poids de Hamming de 3 au maximum.

Pour atteindre encore des performances plus importantes, un additionneur en parallèle a été ajouté, ainsi que des RAM spécifiques. Ainsi, le multiplieur et l'additionneur peuvent fonctionner en parallèle. Par ailleurs, la fréquence du pipeline a été ajustée à 250 MHz, par un travail d'optimisation spécifique à la technologie.

Les résultats obtenus sont inclus dans les tableaux de la section 4.5. Le circuit optimisé économise des blocs DSP et calcule presque 2 fois plus vite un couplage sur BN_{126}, au prix d'une augmentation très importante de la logique (ne pas oublier qu'un slice de Virtex 6 inclut 4 LUT, et correspond donc au double d'un ALM).

FIGURE 4.1 – Rower pour le couplage et pipeline



Chapitre 5

Le RNS : une solution efficace contre les attaques par canaux auxiliaires et par perturbations

Sommaire

5.1 Introduction 82
5.2 Implémentation efficace de la contre-mesure de Bajard et al. [16] 83
5.2.1 La contre-mesure LRA 84
5.2.2 Transfert de mémoire entre Rowers 85
5.2.3 Précalculs 86
5.2.4 Impacts sur l'architecture du Cox-Rower 88
5.3 La contre-mesure de détection de faute pas le RNS (contre-mesure RFD) 88
5.3.1 Modèle de faute 88
5.3.2 Le s-Cox : utilisation d'un développement limité à l'ordre s . . . 89
5.3.3 Détection de faute via l'utilisation du s-Cox 89
5.3.4 Détection d'erreur sur la base $\mathcal{B}'$ 92
5.3.5 Perturbation sur la base $\mathcal{B}$ 93
5.3.6 Perturbation sur le s-Cox ou sur le séquenceur 94
5.3.7 Combinaison des 2 contre-mesures 94
5.4 Implémentation d'un RSA-CRT 1024 protégé avec les contre-mesures LRA et RFD 95
5.4.1 Dimensionnement du Cox-Rower pour l'exécution du RSA-CRT 95
5.4.2 Calcul du représentant de Montgomery 97
5.4.3 Reconstruction du CRT et transformation RNS $\rightarrow$ binaire 98
5.4.4 Latence du RSA-CRT 98
5.5 Conclusion 98
5.6 Annexe A : La contre-mesure de Ciet et al. 100

5.1 Introduction

Comme nous l'avons vu précédemment dans la section 1.3, les attaques par canaux auxiliaires et par perturbation sont devenues, à la suite des travaux de Kocher [86] et

0. Ce chapitre est une traduction d'un article en cours de soumission [62].

Boneh [27] des champs d'investigation importants de la communauté scientifique. Dans ce chapitre, je m'intéresse à la contribution que pourrait fournir le RNS et l'architecture Cox-Rower décrite dans le chapitre 2 au sujet de la résistance à ce type d'attaque.

Parmi l'ensemble des contre-mesures développées dans la littérature pour protéger les primitives asymétriques, seulement 2 contributions utilisent le RNS. La première contribution est celle de Ciet et al dans [37]. Dans cette contribution les auteurs décrivent une implémentation matérielle faible coût du RSA-CRT sur un Cox-Rower. Celle-ci est agrémentée d'une contre-mesure contre les attaques par canaux auxiliaires et une contre-mesure contre les attaques en faute. La deuxième contribution de Bajard et al. [16] propose une contre-mesure par masquage que je décrirai dans la section 5.2. Cette contre-mesure, nommée "Leak Resistant Arithmetic"(LRA) dans la suite de ce chapitre, propose l'utilisation de bases RNS calculées à la volée pour réaliser un masquage des données perturbées. Cette contre-mesure n'a à ce jour jamais été implémentée dans un circuit réel.

Dans ce chapitre, je présente la première implémentation matérielle de la contre-mesure LRA. Je démontre que cette contre-mesure nécessite des adaptations pour être utilisée efficacement dans un Cox-Rower. J'introduis par ailleurs un complément à cette contre-mesure, qui permet de se protéger efficacement contre les attaques par perturbation. Cette contre-mesure permet de détecter pour chaque réduction modulaire qu'une perturbation de la donnée a eu lieu, en utilisant la redondance dans la représentation apportée par une base RNS agrandie. Je démontre l'efficacité de cette contre-mesure que je nomme "RNS Fault Detection" (RFD) dans la suite de ce chapitre. Enfin, je démontre que les 2 contre-mesures peuvent être utilisées ensemble dans un Cox-Rower, en proposant une implémentation de RSA 1024 complètement protégée qui donne le résultat après seulement 0.6 ms sur un FPGA Altera Stratix III. Je démontre le coût faible de ces 2 contre-mesures, l'implémentation complète des 2 contre-mesures pour le RSA-CRT induisant un surplus de 15% en espace et seulement $7,5\%$ en temps.

La lecture du chapitre 3 est un prérequis à la bonne compréhension de cette partie. En effet, les notations ainsi que les concepts utilisés y sont introduits. La section 5.2 décrit l'adaptation de la contre-mesure LRA de Bajard [16], en démontrant qu'une adaptation y est nécessaire afin de profiter pleinement de la puissance du Cox-Rower. Je démontre que cette adaptation ne crée pas d'importante augmentation de la charge de calcul au regard d'un déchiffrement RSA, par exemple. La section 5.3 introduit la contre-mesure RFD, une toute nouvelle contre-mesure donnant au Cox-Rower la capacité de détecter les perturbations à chaque instant du calcul, en créant un minimum de charge supplémentaire, tant au niveau du temps de calcul que du nombre de portes logiques nécessaires à son fonctionnement. Enfin, dans la section 5.4, j'exhibe une implémentation du RSA-CRT 1024 bits sur un unique Cox-Rower, où j'implémente les 2 contre-mesures. Je démontre ainsi, par cette implémentation, le faible coût des contre-mesures au regard de la primitive RSA. En annexe 5.6, je propose une discussion sur une proposition de Ciet et Quisquater [37], dans laquelle les auteurs proposent d'utiliser le RNS comme contre-mesure contre les canaux auxiliaires et les fautes.

5.2 Implémentation efficace de la contre-mesure de Bajard et al. [16]

Dans cette section, je démontre que la contre-mesure Leak Resistant Arithmetic (dont je rappelle les principes dans la section 5.2.1) est parfaitement implémentable sur un Cox-Rower. S'il ne faisait aucun doute qu'il est possible d'utiliser la contre-mesure sur un

Cox-Rower, la difficulté réside dans le fait que pour cette contre-mesure soit efficace, les bases RNS doivent pouvoir être changées le plus souvent possible. Ainsi, je propose de la changer systématiquement à chaque début de calcul. Or, nous avons vu dans le chapitre 2 que de nombreux précalculs sont nécessaires pour réaliser l'algorithme d'exponentiation, et ces précalculs dépendent de manière importante de la valeur des bases RNS, il est donc impossible de stocker l'ensemble de ces valeurs sans limiter le nombre de bases possibles. Dans cette partie, je démontre qu'à partir des mêmes précalculs indépendants des bases que Bajard et al. proposent dans leur contribution [16], je peux reconstruire l'intégralité des éléments nécessaires à l'algorithme de réduction de Montgomery, ainsi qu'aux transformations binaire $\rightarrow$ RNS, et sans utiliser de ressource matérielle supplémentaire que le Cox-Rower.

5.2.1 La contre-mesure LRA

Considérons p un grand nombre premier et une base RNS $\mathcal{B} = (m_1; \cdots; m_{2n})$, de telle manière que n'importe quelle partition de $\mathcal{B}$ en 2 bases RNS de taille n $\mathcal{B}$ et $\mathcal{B}'$ réponde aux critères définis par le théorème 2 pour la réduction de Montgomery. Ces conditions sont aisées à obtenir, en employant des pseudo-Mersenne de même taille r, comme définis dans la section 2.4. L'idée de la contre-mesure est de tirer aléatoirement une permutation γ de $[1; 2n]$ vers lui-même. Cette permutation permet d'obtenir 2 bases RNS notées $\mathcal{B}_{1,\gamma} = (m_{\gamma(1)}; \cdots; m_{\gamma(n)})$ et $\mathcal{B}_{2,\gamma} = (m_{\gamma(n+1)}; \cdots; m_{\gamma(2n)})$.

Le choix aléatoire de base RNS crée ainsi automatiquement un masquage lors du passage au représentant de Montgomery d'un élément. Ainsi, le représentant de Montgomery de la valeur X est $|X\mathcal{M}(\mathcal{B}_{1,\gamma})|_p$, qui est unique pour chaque ensemble $\{m_{\gamma(1)}; \cdots; m_{\gamma(n)}\}$. Par conséquent, le nombre de masques possibles est égal à $\mathbf{C}_n^{2n}$, soit asymptotiquement à $2^{2n}/\sqrt{\pi n}$ (valeur asymptotique atteinte même pour des tailles cryptographiques). Comme nous le verrons dans la partie 5.4 où j'exhibe une implémentation Cox-Rower, où $n = 15$ et $r = 36$ (suffisant pour réaliser un RSA-CRT 1024 bit), cela représente 2^{27} masques différents. En outre, chacun des canaux étant affecté à un Rower, si l'on considère que la fuite issue d'un Rower n'est pas identique à un autre (c'est en particulier vrai dans le cas de DEMA où le signal peut être couplé à un Rower en particulier), alors la fuite dépendra non seulement de l'ensemble $\{m_{\gamma(1)}; \cdots; m_{\gamma(n)}\}$, mais aussi de l'affectation de chacun des canaux sur un Rower. Ceci augmente encore le nombre de masques possibles.

L'intérêt de ce type de masque sur un masque multiplicatif classique, est que l'étape de démasquage ne nécessite pas d'inversion. En effet, celui-ci s'effectue grâce à une simple réduction de Montgomery.

Dans leur contribution [16], les auteurs parviennent à limiter le nombre de précalculs nécessaires à l'élaboration de nouvelles bases à $4n^2 - 2n$, toutes ces valeurs étant indépendantes de γ, puisqu'il s'agit de $\mu_{i,j} = |{m_i}^{-1}|_{m_j}$. Afin d'obtenir ce résultat, les auteurs privilégient l'utilisation du MRS (algorithme 12) pour réaliser le changement de base. En effet, on peut constater que les seuls éléments précalculés nécessaires à ce type de changement de base sont les valeurs $\mu_{i,j}$. De manière identique, l'utilisation d'un schéma de Horner pour la transformation MRS vers RNS ne nécessite pas d'autres précalculs que $|m_i|_{m_j}$, qui dans les calculs modulaires peuvent être gardés sous la forme m_i.

Par ailleurs, les valeurs nécessaires à la transformation de Montgomery, telles que $(|-p^{-1}|_{\mathcal{M}(\mathcal{B}_{1,\gamma})})_{\mathcal{B}_{1,\gamma}}$ ou $(p)_{\mathcal{B}_{2,\gamma}}$ ne sont en fait rien d'autre que les valeurs $|-p^{-1}|_{m_{\gamma(i)}}$ et $|p|_{m_{\gamma(n+i)}}$, et ne dépendent pas de γ.

Enfin, la seule difficulté réelle de cette contre-mesure est le calcul des valeurs $|\mathcal{M}(\mathcal{B}_{1,\gamma})^2|_p$ et $|\mathcal{M}(\mathcal{B}_{1,\gamma})|_p$ (la deuxième étant le représentant de Montgomery de 1, elle peut être utile

TABLE 5.1 – Liste des précalculs à transmettre pour un canal m_i de $\mathcal{M}(\mathcal{B})$

Nom	valeur	nombre
$\mathbf{i}$	$\|-1\|_{m_i}$	1
$m_{j,i}$	$\|m_j\|_{m_i}$	$2n-1$
$\mu_{j,i}$	$\|{m_j}^{-1}\|_{m_i}$	$2n-1$
π_i	$\|-p^{-1}\|_{m_i}$	1
p_i	$\|p\|_{m_i}$	1
Mp_i	$\|\|\mathcal{M}(\mathcal{B})\|_p\|_{m_i}$	1
MMp_i	$\|\|\mathcal{M}(\mathcal{B})^2\|_p\|_{m_i}$	1
total		$4n+3$

dans certains algorithmes d'exponentiation comme le MSB, LSB ou Montgomery ladder). Ces valeurs sont nécessaires pour le calcul du représentant de Montgomery. Bajard et al. découvrent en réalité qu'on peut aisément échanger les bases $\mathcal{B}_{1,\gamma}$ et $\mathcal{B}_{2,\gamma}$ pour obtenir :

$$(|\mathcal{M}(\mathcal{B}_{1,\gamma})^2|_p)_{\mathcal{B}_1\cup\mathcal{B}_2} = \texttt{RedM}(|\mathcal{M}(\mathcal{B})^2|_p, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma}) \text{ (algorithme 11).} \tag{5.1}$$

Ainsi, $|\mathcal{M}(\mathcal{B}_{1,\gamma})^2|_p$ peut être obtenue après une simple réduction de Montgomery d'une valeur indépendante de γ, ce qui ne nécessite aucun précalcul.

Jusqu'à ma contribution [62], aucune implémentation de cette contre-mesure n'avait été réalisée à ma connaissance. En effet, l'inconvénient de la contre-mesure telle qu'elle est décrite dans [16] est qu'elle impose l'utilisation de la transformation MRS de Szabo et Tanaka [116]. Or, comme nous l'avons vu dans le chapitre 3, il existe d'autres transformations, plus efficaces, mais gourmandes en précalcul. En particulier, le MRS empêche une implémentation efficace dans une architecture parallèle, à cause du calcul du mot de poids fort, qui dépend de tous les mots précédents (voir chapitre 3). Ainsi, l'utilisation dans un Cox-Rower de ce changement de base coûterait au minimum $2n$ cycles sans compter les attentes de pipeline. Comparée à l'algorithme de Kawamura, cette contre-mesure est chère.

Dans les 2 sections suivantes, je démontre qu'il est possible, et même extrêmement efficace de réaliser les précalculs pour la transformation de Kawamura (algorithme 13), et donc de se passer de la transformation MRS, à la condition d'avoir à disposition un Cox-Rower. En effet, à partir d'un ensemble de précalculs indépendants de γ (en fait, il s'agit de ceux définis par Bajard et al.), il est possible de reconstruire l'ensemble des précalculs contenus dans le tableau 2.1.

Le traitement se réalise en 2 phases : une étape de transferts de précalculs vers les mémoires principales de chaque Rower, puis une étape de calcul dans les bases RNS.

5.2.2 Transfert de mémoire entre Rowers

Supposons la donnée d'un Rower composé de n/ν Rower. La première étape consiste à assigner chaque canal m_i de $\mathcal{B}$ à un Rower. Chaque Rower se voit donc assigner ν canaux de la base $\mathcal{B}_{1,\gamma}$, et ν de la base $\mathcal{B}_{2,\gamma}$. Ensuite, pour chaque canal m_x assigné au Rower j, un transfert doit être réalisé entre une mémoire contenant les éléments précalculés de chaque canal vers sa mémoire principale.

La liste des éléments précalculés à transférer vers les mémoires est décrite dans le tableau 5.1. Il faut noter que les éléments $m_{i,i}$ et $\mu_{i,i}$ ne sont pas comptés : par définition, ces valeurs sont assignées à 1.

Le transfert peut être réalisé séquentiellement sur chaque mémoire de Rower. Dans ce cas, le coût en cycles est de $8n^2 + 6n$ (soit 1881 cycles dans le cas de l'implémentation de la section 5.4). En revanche le coût en matériel est nul, puisque les transferts peuvent être réalisés de la mémoire principale d'un Rower à un autre, via l'utilisation du registre d'entrée/sortie de chaque Rower. La condition nécessaire est qu'il existe dans la mémoire principale, pour tout j une adresse où la donnée présente est égale à 1 pour le Rower j, et 0 pour les autres Rower. L'adresse de stockage de cette valeur est notée 1_j.

Supposons $u_1, u_{n/\nu}$ stockées à l'adresse $@_1, @_{n/\nu}$ de la mémoire principale de chaque Rower. Soit une permutation ζ de $[1; \cdots; n/\nu]$. Je souhaite réaliser l'écriture à la même adresse $@_d$ des valeurs $u_{\zeta(1)}, \cdots u_{\zeta(n/\nu)}$ sur chaque Rower. L'objectif peut être atteint avec le Cox-Rower décrit dans le chapitre 3 grâce au traitement suivant :

1. cycle 0 : le Rower 1 charge la valeur u_1 dans le registre Reg_S,
2. cycle 1 : la données u_1 est transmise via le bus à tous les Rower et multipliée par la valeur $1_{\zeta(1)}$. Le résultat est accumulé. La donnée u_2 est stockée par le Rower 2 dans son registre reg_S,
3. la données u_2 est transmise via le bus à tous les Rower et multipliée par la valeur $1_{\zeta(1)}$. Le résultat est accumulé. La donnée u_3 est stockée par le Rower 2 dans son registre reg_S,
4. ...
5. cycle n/nu : La donnée u_n/ν est accumulée et le résultat est écrit dans la mémoire $@_d$ sur l'ensemble des Rower. En parallèle, il est possible de réaliser le cycle 0 du transfert suivant.

Comme indiqué, le traitement repose entièrement sur un cadencement par le séquenceur (via le pilotage de l'adresse $@_i$). Aucun matériel supplémentaire n'est nécessaire pour réaliser les transferts de mémoire entre les Rowers. Si ce mode de transfert semble relativement cher, il reste néanmoins négligeable par rapport au temps total de traitement de l'exponentiation RSA (voir section 5.4).

Un autre mode de transfert permet d'adresser parallèlement toutes les mémoires, ce qui permet de réduire à $8n\nu + 6\nu$ le nombre de cycles nécessaires. Pour cela, du matériel doit être ajouté afin de permettre le transfert des données simultanément. Ce transfert peut être réalisé toujours spécifiquement entre les mémoires elles-mêmes. Un élément d'architecture matérielle permettant de réaliser les permutations aléatoires en parallèle, comme un réseau de Benz (voir figure 5.1) peut s'avérer alors utile pour réaliser en parallèle les transferts d'une mémoire à l'autre. Ce réseau permet le transfert entre les entrées et sorties de chaque mémoire. Afin de s'avérer efficace pour la suite du traitement, la configuration du réseau doit être couplée au séquenceur afin d'obtenir un adressage cohérent dans les mémoires des Rower. Cette possibilité n'a pas été implémentée dans les résultats fournis dans la section 5.4.

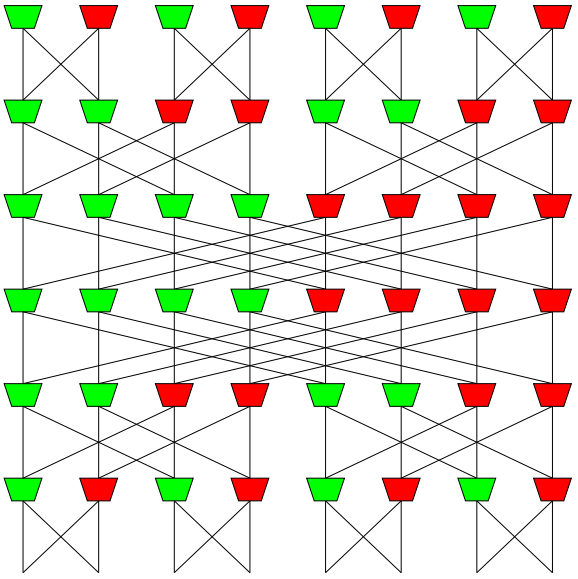
5.2.3 Précalculs

La deuxième étape de préparation consiste à calculer les valeurs contenues dans le tableau 2.1 à partir des précalculs du tableau 5.1.

En premier lieu, les valeurs suivantes sont calculées, pour un coût total de $8n\nu$ cycles

$$
\begin{array}{lcccl}
(A)_{\mathcal{B}_{1,\gamma}} & = & (\mathcal{M}(\mathcal{B}_{2,\gamma}))_{\mathcal{B}_{1,\gamma}} & = & \{|\prod_{i=n}^{2n} m_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(j)}}\} \text{ pour } j \in [1,n] \\
(B)_{\mathcal{B}_{1,\gamma}} & = & (\mathcal{M}(\mathcal{B}_{2,\gamma})^{-1})_{\mathcal{B}_{1,\gamma}} & = & \{|\prod_{i=n}^{2n} \mu_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(j)}}\} \text{ pour } j \in [1,n] \\
(C)_{\mathcal{B}_{1,\gamma}} & = & \{|\mathcal{M}(\mathcal{B}_{1,\gamma})/m_{\gamma(j)}|_{m_{\gamma(j)}}\} & = & \{|\prod_{i=0}^{n} m_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(j)}}\} \text{ pour } j \in [1,n] \\
(D)_{\mathcal{B}_{1,\gamma}} & = & \{|m_{\gamma(j)}/\mathcal{M}(\mathcal{B}_{1,\gamma})|_{m_{\gamma(j)}}\} & = & \{|\prod_{i=0}^{n} \mu_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(j)}}\} \text{ pour } j \in [1,n]
\end{array}
$$

FIGURE 5.1 – Réseau de Benz $n = 8$



et de manière symétrique en échangeant $\mathcal{B}_{1,\gamma}$ et $\mathcal{B}_{2,\gamma}$:

$$\begin{array}{lcccl}
(A')_{\mathcal{B}_{2,\gamma}} & = & (\mathcal{M}(\mathcal{B}_{1,\gamma}))_{\mathcal{B}_{2,\gamma}} & = & \{|\prod_{i=0}^{n} m_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(n+j)}}\} \text{ for } j \in [1,n] \\
(B')_{\mathcal{B}_{2,\gamma}} & = & (\mathcal{M}(\mathcal{B}_{1,\gamma})^{-1})_{\mathcal{B}_{2,\gamma}} & = & \{|\prod_{i=0}^{n} \mu_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(n+j)}}\} \text{ pour } j \in [1,n] \\
(C')_{\mathcal{B}_{2,\gamma}} & = & \{|\mathcal{M}(\mathcal{B}_{2,\gamma})/m_{\gamma(n+j)}|_{m_{\gamma(n+j)}}\} & = & \{|\prod_{i=n}^{2n} m_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(n+j)}}\} \text{ pour } j \in [1,n] \\
(D')_{\mathcal{B}_{2,\gamma}} & = & \{|m_{\gamma(n+j)}/\mathcal{M}(\mathcal{B}_{2,\gamma})|_{m_{\gamma(n+j)}}\} & = & \{|\prod_{i=n}^{2n} \mu_{\gamma(i),\gamma(n+j)}|_{m_{\gamma(n+j)}}\} \text{ pour} j \in [1,n]
\end{array}$$

Enfin, les éléments nécessaires à la réduction de Montgomery $\texttt{RedM}(X, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$ du tableau 2.1 peuvent être aisément obtenus de la manière suivante :

$$\begin{array}{lcccl}
P_{i,\mathcal{B}_{1,\gamma}} & = & (\quad -p^{-1} \times A \quad)_{\mathcal{B}_{1,\gamma}} & \nu \text{ cycles} \\
M_{i,j,\mathcal{B}_{1,\gamma}\to\mathcal{B}_{2,\gamma}} & = & (\quad p \times m_{\gamma(i)} \times D' \quad)_{\mathcal{B}_{2,\gamma}} & (n+1)\nu \text{ cycles} \\
C_{i,\mathcal{B}_{1,\gamma}\to\mathcal{B}_{2,\gamma}} & = & (\quad -p \times m_{\gamma(n+j)} \times D' \quad)_{\mathcal{B}_{2,\gamma}} & 2\nu \text{ cycles} \\
P_{i,\mathcal{B}_{2,\gamma}} & = & (\quad B' \times D' \quad)_{\mathcal{B}_{2,\gamma}} & \nu \text{ cycles} \\
Q_{i,\mathcal{B}_{2,\gamma}} & = & (\quad C' \quad)_{\mathcal{B}_{2,\gamma}} & 0 \text{ cycles} \\
M_{i,j,\mathcal{B}_{2,\gamma}\to\mathcal{B}_{1,\gamma}} & = & (\quad A \times \ \mu_{\gamma(n+j),\gamma(i)} \quad)_{\mathcal{B}_{1,\gamma}} & \nu n \text{ cycles} \\
C_{i,\mathcal{B}_{2,\gamma}\to\mathcal{B}_{1,\gamma}} & = & (\quad -A \quad)_{\mathcal{B}_{1,\gamma}} & \nu \text{ cycles}
\end{array}$$

et de manière symétrique pour les précalculs nécessaires à $\texttt{RedM}(X, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$:

$$\begin{array}{lcccl}
P'_{i,\mathcal{B}_{2,\gamma}} & = & (\quad -p^{-1} \times A' \quad)_{\mathcal{B}_{2,\gamma}} & \nu \text{ cycles} \\
M'_{i,j,\mathcal{B}_{2,\gamma}\to\mathcal{B}_{1,\gamma}} & = & (\quad p \times m_{\gamma(n+i)} \times D \quad)_{\mathcal{B}_{1,\gamma}} & (n+1)\nu \text{ cycles} \\
C'_{i,\mathcal{B}_{1,\gamma}\to\mathcal{B}_{2,\gamma}} & = & (\quad -p \times m_{\gamma(j)} \times D \quad)_{\mathcal{B}_{1,\gamma}} & 2\nu \text{ cycles} \\
P'_{i,\mathcal{B}_{2,\gamma}} & = & (\quad B \times D \quad)_{\mathcal{B}_{1,\gamma}} & \nu \text{ cycles} \\
Q'_{i,\mathcal{B}_{2,\gamma}} & = & (\quad C \quad)_{\mathcal{B}_{1,\gamma}} & 0 \text{ cycles} \\
M'_{i,j,\mathcal{B}_{2,\gamma}\to\mathcal{B}_{1,\gamma}} & = & (\quad A' \times \ \mu_{\gamma(n+j),\gamma(i)} \quad)_{\mathcal{B}_{2,\gamma}} & \nu n \text{ cycles} \\
C'_{i,\mathcal{B}_{2,\gamma}\to\mathcal{B}_{1,\gamma}} & = & (\quad -A' \quad)_{\mathcal{B}_{2,\gamma}} & \nu \text{ cycles}
\end{array}$$

On peut constater que dans la mesure où l'ensemble de ces calculs se réalisent directement dans les bases RNS, cette étape de précalcul bénéficie complètement de l'architecture

du Cox-Rower. Ainsi, cette opération de précalcul est effectuée en $(4n+12)\nu$ cycles. A cela, il est nécessaire d'ajouter $2(\nu+3)$ cycles s'il faut ajouter un deuxième module, comme c'est le cas avec le RSA en mode CRT (p et q).

Il est important de noter que la valeur du nombre p, qui se retrouve dans les calculs de $P_{i,\mathcal{B}_{2,\gamma}}$,$M_{i,j,\mathcal{B}_{2,\gamma}\rightarrow\mathcal{B}_{1,\gamma}}$ et $C_{i,\mathcal{B}_{1,\gamma}\rightarrow\mathcal{B}_{2,\gamma}}$, est systématiquement masquée par γ durant cette phase. Cela est utile, notamment pour le RSA-CRT, où p correspond à la clé privée. Ainsi, l'attaquant ne pourra pas réaliser une DPA classique sur la valeur de p durant cette phase.

5.2.4 Impacts sur l'architecture du Cox-Rower

Le seul impact sur l'architecture est la nécessité d'ajouter une capacité de chargement aux mémoires ROM_1 et ROM_2 (sans la contremesure, ces mémoires sont de simples ROM). A noter que leur capacité totale est de 2ν mots de taille maximale $r/2$ bits dans le cas de ROM_1, et de $\nu(1+\pi)$ mots de r bits dans le cas de ROM_2 (où π est la quantité de nombres premiers susceptibles d'être traités simultanément par le Cox-Rower). Ainsi, une implémentation sous forme de registres n'est pas pénalisante. La capacité de transfert peut aisément être insérée dans l'architecture, pour permettre le transfert depuis la mémoire principale. Le coût du transfert vers ces mémoires est $\nu(3+\pi)$ cycles.

5.3 La contre-mesure de détection de faute pas le RNS (contre-mesure RFD)

Dans cette section, j'introduis une nouvelle contre-mesure contre les attaques par perturbations présentée dans la section 1.3.2, que j'appelle la contre-mesure RFD. Cette contre-mesure a été introduite dans ma contribution [62]. De la même manière que la contre-mesure LRA, cette contre-mesure s'applique sur l'ensemble des algorithmes traitant des calculs modulo des grands nombres.

5.3.1 Modèle de faute

Je considère un Cox-Rower défini par 2 bases $\mathcal{B} = (m_1;\cdots;m_n)$ et $\mathcal{B}' = (m'_1;\cdots;m'_n)$, non nécessairement aléatoires, et connues de l'attaquant. Tous les éléments m_i et m'_i sont des pseudo-Mersenne de taille r et je définis la valeur $r_\varepsilon = Max(\lceil log_2(2^r - m_i)\rceil, \lceil log_2(2^r - m'_i)\rceil$ pour $i \in [1,n]$). Soit p le grand nombre premier avec $\mathcal{M}(\mathcal{B}\cup\mathcal{B}')$. Le nombre de Rowers disponibles est défini par le rapport n/ν.

Le modèle de faute adressé est une perturbation sur un simple Rower. Cette perturbation peut être aléatoire ou bien spécifique à un bit en particulier. Ce type de faute est typique d'une attaque laser, qui permet de contrôler le résultat de la perturbation à la condition de réduire le rayon du faisceau. Au cours de cette section, je vais démontrer que la contre-mesure proposée permet de détecter complètement ce type de faute. L'objectif de la contre-mesure est de permettre de protéger les Rowers contre ce type d'attaque, dans la mesure où ceux-ci forment l'essentiel du hardware d'un Cox-Rower. Une attaque sur le Cox ou le séquenceur n'est pas couverte par cette contre-mesure, néanmoins leur taille raisonnable permet l'utilisation de contre-mesures simples à mettre en place, comme la redondance, par exemple.

L'intuition que je suis pour le dessin de cette contre-mesure est que si la taille des bases RNS est plus importante (e.g en ajoutant un élément de plus que nécessaire à $\mathcal{B}$ et $\mathcal{B}'$), tandis que la réduction de Montgomery garantit un résultat entre 0 and $3p$, je peux utiliser le Rower ajouté comme preuve qu'aucune faute n'a été commise. En outre, il est facile de

démontrer que si les représentations RNS de 2 valeurs X et Y sont proches l'une de l'autre (i.e tous les chiffres RNS sont égaux sauf 1), alors elles ne le sont pas dans la représentation multiprécision : la valeur absolue de leur différence est grande. Ceci est par ailleurs connu depuis les travaux de Szabo et Tanaka [116]. L'objectif est d'exploiter efficacement cette propriété, sans réduire les capacités du Cox-Rower à effectuer efficacement l'arithmétique dans $\mathbb{Z}/p\mathbb{Z}$. Cela impose d'examiner le comportement de l'algorithme de Kawamura (algorithme 13) en présence de perturbations.

5.3.2 Le s-Cox : utilisation d'un développement limité à l'ordre s

Comme nous le verrons par la suite, la contre-mesure proposée nécessite un changement important au niveau du Cox, puisque celui-ci sera en charge de la détection de la perturbation. Pour cela, il est nécessaire de rendre la fonction d'approximation des valeurs (ξ_i/m_i) plus précise (voir section 2.4.2).

Je dois tout d'abord redéfinir la fonction $eval(\xi_i, m_i)$ définie dans la section 2.3.4. Cette fonction est nécessaire à la ligne 9 de l'algorithme 13. En effet la proposition de Kawamura $trunc_l(\xi_i)$ ne permet pas une approximation suffisante pour permettre une détection de faute efficace. Je propose de la remplacer par un développement limité à l'ordre s.

$$\xi_i/m_i \simeq \xi_i(\sum_{j=0}^{s-1} \frac{(2^r - m_i)^j}{2^{r(j+1)}}) \tag{5.2}$$

Ainsi, l'approximation $eval(\xi_i, m_i)$ reste une sous-estimation comme l'avait proposé Kawamura et la valeur de $errmax$ definie dans la section 2.3.4 respecte l'inégalité suivante :

$$errmax \leq \sum_{i=1}^{n} \left(\frac{2^r - m_i}{2^r}\right)^s \leq 2^{-s(r-r_\varepsilon)+\lceil log_2(n)\rceil} \tag{5.3}$$

Naturellement, le calcul de l'équation 5.2 doit être réalisé via une architecture de Cox revisitée. La définition des pipelines des chapitres 3 et 4, a permis de déterminer que le signal SC doit être levé 2 cycles après le passage de la donnée ξ_i sur le bus principal pour une utilisation efficace dans le pipeline des Rowers. Cette contrainte peut être levée au prix d'un nouveau dessin de ces pipelines (et le risque d'ajouter un étage supplémentaire avant l'accumulation, ce qui peut être pénalisant pour la taille du circuit complet). Le choix a été fait de privilégier la redéfinition du Cox plutôt que celui du pipeline des Rowers. Je propose donc 2 implémentations différentes, la première (figure 5.2) réalisant le calcul de l'équation 5.2 en 2 cycles, la deuxième (figure 5.3) relâchant cette contrainte. Expérimentalement, j'ai constaté que pour $s = 2$, la première est à privilégier car elle ne limite pas la fréquence maximale du Cox-Rower. Pour des valeurs de s supérieures, le nombre d'additions à réaliser à l'étage de pipeline 2 est trop important et limite la fréquence maximale, il faut donc privilégier la deuxième architecture, qui permet d'assurer le temps nécessaire pour recomposer l'addition.

On peut constater que le plus gros multiplieur est de taille $r \times (s-1)r_\varepsilon$. Par conséquent, il est nécessaire de conserver $(s-1)r_\varepsilon$ à une taille permettant l'implémentation de la multiplication en 1 cycle. Par exemple $s < \lceil r/r_\varepsilon \rceil$ garantit que le plus gros multiplieur sera plus petit que le premier étage du Rower (et ainsi son chemin critique).

5.3.3 Détection de faute via l'utilisation du s-Cox

Supposons maintenant que $\mathcal{B}'$ soit supérieur à $3p$, la valeur maximale que peut prendre $(S)_{\mathcal{B}'}$ à la ligne 4 de l'algorithme 11. Je définis σ tel que $\mathcal{M}(\mathcal{B}') > 2^{\sigma+1}3p$. Je suppose toujours $\mathcal{M}(\mathcal{B})$ supérieur à αp. Je peux démontrer le théorème suivant :

FIGURE 5.2 – Architecture de Cox à 2 étages

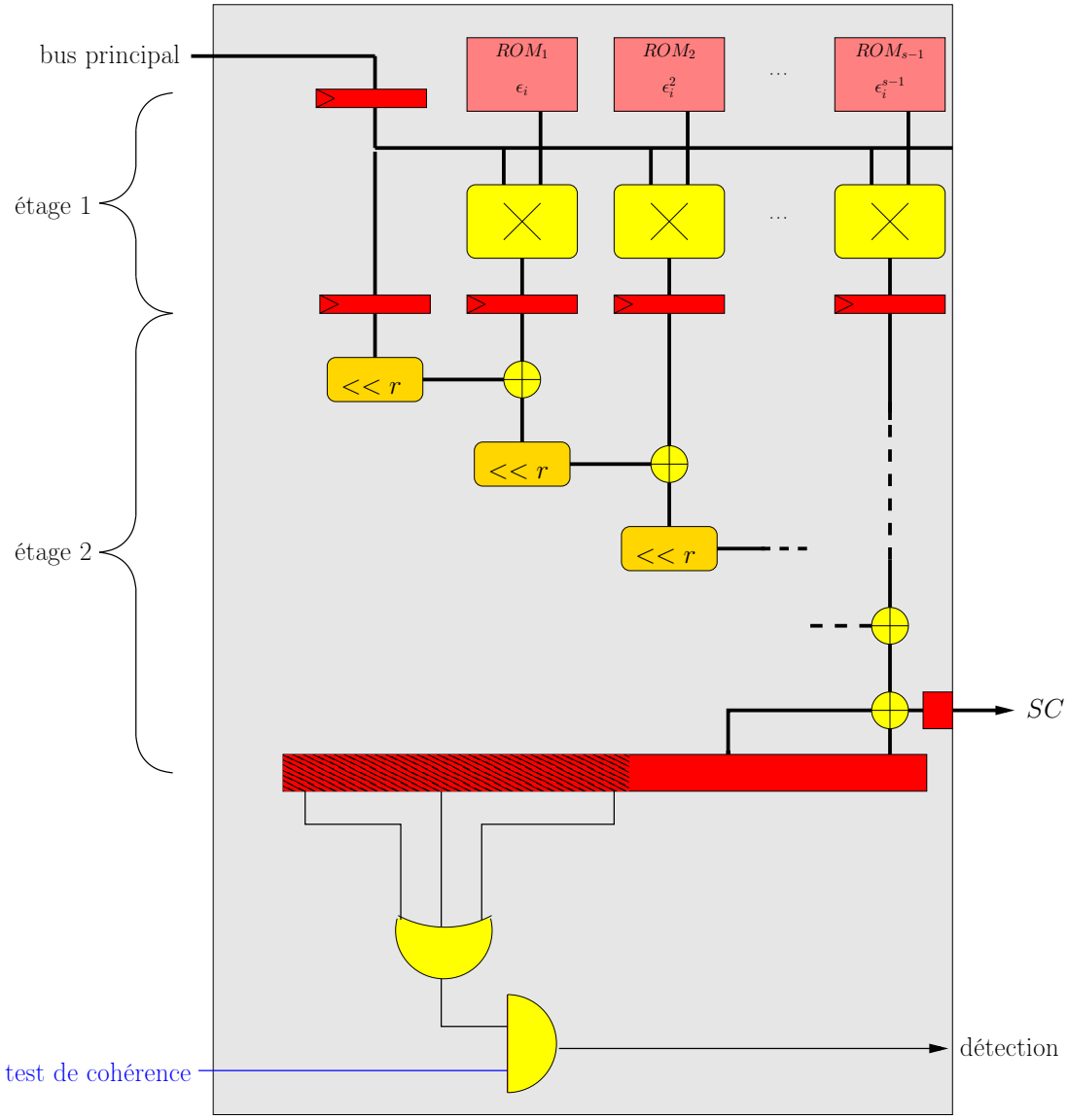
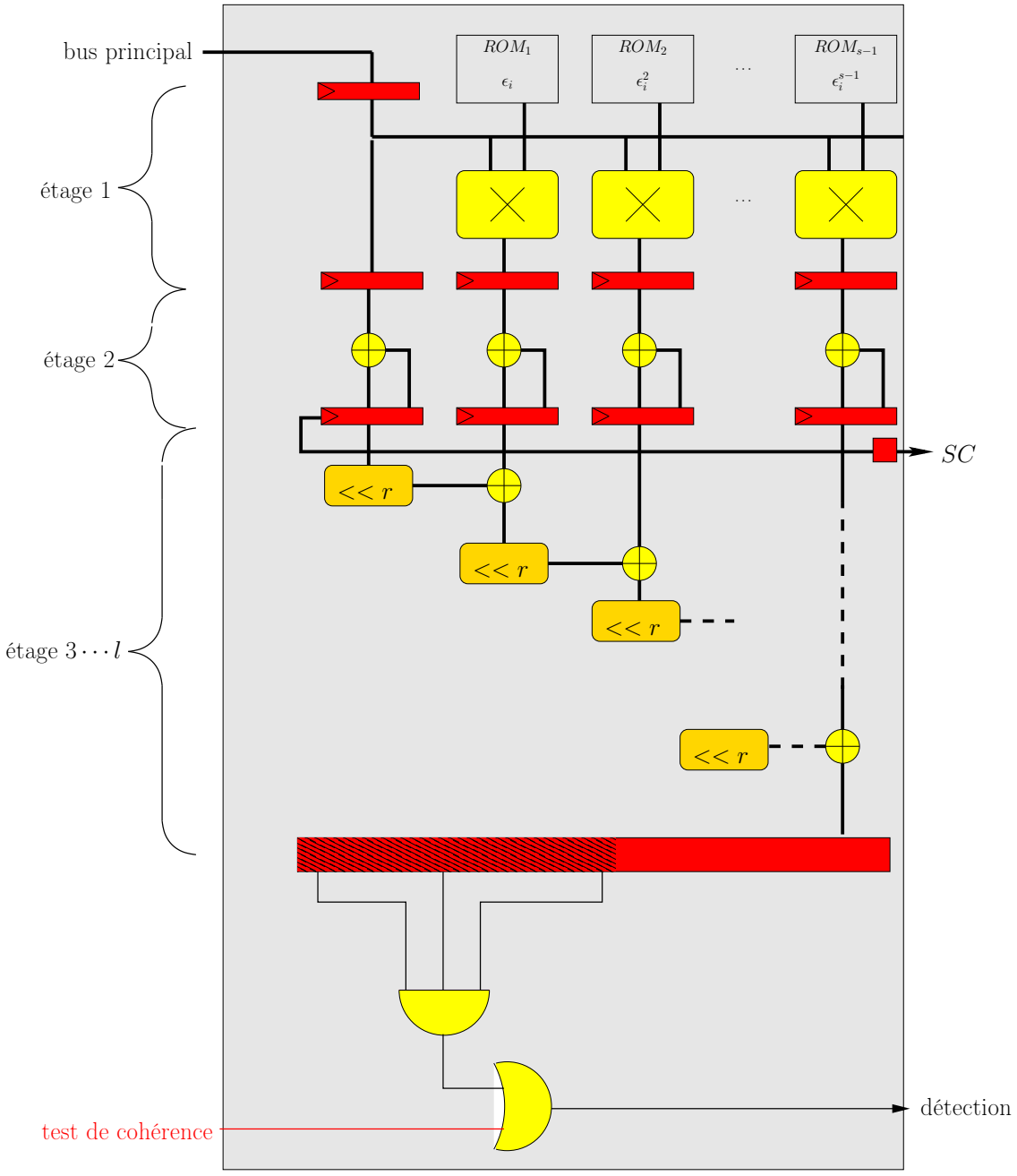


FIGURE 5.3 – Architecture de Cox à $l > 2$ étages



Théorème 11 *Considérons les 3 conditions suivantes :*
- *le niveau s du développement limité proposé par le Cox respecte $s(r-r_\varepsilon) \geq \lceil log_2(n) \rceil + \sigma + 1$,*
- *$errinit = 2^{-\sigma-1}$*
- *L'entrée X de l'algorithme de Montgomery est inférieure à αp^2,*

Dans ce cas, les σ bits significatifs de l'accumulateur du Cox sont égaux à 0 à la fin du second changement de base (ligne 5 de l'algorithme 11).

Preuve :

Comme $X \leq \alpha p^2$, on déduit que $S' \leq 3p$ à la ligne 4 de l'algorithme 11 (théorème 2). Ainsi dans l'exécution de la ligne 5 de l'algorithme 13, on peut déduire :

$$0 \leq \texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m'_i}\right) = \frac{S'}{\mathcal{M}(\mathcal{B}_1)} < 2^{-\sigma-1}, \tag{5.4}$$

où $\texttt{dec}(x)$ désigne $x - \lfloor x \rfloor$, soit la partie décimale de x. Par ailleurs, avec la définition de s et $errinit$, on peut déduire qu'à la fin de l'algorithme 13 :

$$\texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m_i}\right) \leq reg\_accumulator \leq \texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m_i}\right) + errinit \tag{5.5}$$

Ainsi, on en déduit l'inéquation finale suivante : $0 \leq reg\_accumulator < 2^{-\sigma}$ CQFD.

Ainsi, en la présence d'aucune perturbation, quelque soit le traitement modulaire réalisé, le signal de détection présent dans la figure 5.2 et 5.3 reste au niveau bas. Dans la suite de cette section, je propose d'introduire des fautes dans certaines variables du Rower.

5.3.4 Détection d'erreur sur la base $\mathcal{B}'$

Dans cette section j'introduis une erreur sur une valeur manipulée dans $\mathcal{B}'$. Cela peut correspondre à une perturbation sur $(X)_{\mathcal{B}'}$, aux lignes 3 et 4 de l'algorithme 11, à la ligne 5, 11 et 13 du premier changement de base, la ligne 2 du second changement de base ou sur n'importe quel précalcul sur $\mathcal{B}'$.

Je considère s, $\mathcal{M}(\mathcal{B}_2)$ et X une valeur non perturbée telles que les conditions du théorème 11 soient vérifiées. Je peux en déduire le théorème suivant :

Théorème 12 *Si $\sigma \geq r + 1$, toute faute ϵ sur un des canaux de $\mathcal{B}'$ sera détectée par le s-Cox ou sera automatiquement corrigée par l'algorithme de réduction.*

Preuve : On considère une faute unique sur le canal i de la base $\mathcal{B}'$. Le traitement par l'algorithme de réduction conduit automatiquement à $\xi'_i = \xi_i + \epsilon_i$ et $\xi'_j = \xi_j$ pour tout $j \neq i$ à la ligne 2 du second changement de base. Les registres du Cox-Rower étant de taille r, on a : $-m'_i < -\xi_i \leq \epsilon_i \leq 2^r - \xi_i < 2^r$. Suite à la faute, on obtient alors dans l'accumulateur du Cox la valeur suivante :

$$reg\_accumulateur' = \texttt{dec}(reg\_accumulator + eval(\epsilon_i, m'_i))$$

2 cas sont alors à considérer :

– $\epsilon_i = m'_i$: on voit aisément que cette faute n'a pas d'impact sur le résultat : en effet, l'effet induit sur $(S)_B$ (ligne 11 de l'algorithme 13) est $\mathcal{M}(\mathcal{B}_2)$ et est automatiquement corrigé par la ligne 16, qui est exécutée une fois de trop par rapport à l'exécution non perturbée. Il est possible cependant que la détection soit activée, en fonction de l'approximation, néanmoins un défaut d'exécution ne change pas le résultat final.
– autres ϵ_i : suivant l'inéquation suivante,

$$\frac{\epsilon_i}{2^r} < eval(\epsilon_i, m'_i)) < \epsilon_i \left(\frac{1}{2^r} + \frac{1}{2^{r+1}} \right). \tag{5.6}$$

On peut constater que pour n'importe quelle valeur de $\epsilon_i \notin \{0; m'_i\}$, `dec`$(reg\_accumulator + eval(\epsilon_i, m'_i))$ possède au moins 1 bit dans les σ poids forts à 1. Ainsi, ce bit sera détecté par le s-Cox.

5.3.5 Perturbation sur la base $\mathcal{B}$

De la même manière, j'introduis maintenant une perturbation à la ligne 1 de l'algorithme 11, à la ligne 2 du premier changement de base ou aux lignes 11 et 13 du deuxième changement de base (ou sur n'importe quelle valeur précalulée de $\mathcal{B}$). Toutes ces fautes induisent une valeur incorrecte ξ_i à la ligne 2 du premier changement de base (la condition nécessaire est que l'algorithme de réduction de base soit exécuté sur la valeur perturbée. Si ce n'est pas le cas, nous verrons dans la section suivante que l'on peut alors utiliser la transformation vers la représentation binaire. En effet, la valeur attendue par cette transformation doit être inférieure à $3p$. Si ce n'est pas le cas, une perturbation a été détectée).

Théorème 13 *Considérons les 3 conditions suivantes*
– *s le niveau du développement limité du Cox est tel que $s\rho_\varepsilon \geq \lceil log_2(n) \rceil + \sigma + 1$,*
– *$errinit = 2^{-\sigma-1}$*
– *X l'entrée de l'algorithme* `RedM` *est inférieure à αp^2,*
alors, pour ϕ tel que $\sigma \geq (r + \lceil log(6n) \rceil + 1) + \phi$, la probabilité qu'il existe une valeur précalulée et une perturbation sur une seule valeur de $\mathcal{B}$ qui ne soit pas détectée par le $s-Cox$ vérifie l'inéquation :

$$Pr(\exists f \text{ non détecté }) < 2^{-\phi} \tag{5.7}$$

Preuve :

On considère une faute unique ϵ_i, avec $\xi'_i = \epsilon_i + \xi_i$ et pour $j \neq i$, $\xi'_j = \xi_j$ au premier changement de base. Regardons comment cette faute est propagée dans l'algorithme de réduction de Montgomery.
– $Q'' = |Q' + \epsilon_i \mathcal{M}(\mathcal{B}_1)/m_i|_{\mathcal{M}(\mathcal{B}_2)}$ si la ligne 13 de l'algorithme 13 a été exécutée autant de fois que pour les valeurs non perturbées,
– $Q'' = |Q' + \varepsilon\mathcal{M}(\mathcal{B}_1)/m_i - \mathcal{M}(\mathcal{B}_1)|_{\mathcal{M}(\mathcal{B}_2)}$ si la ligne 13 a été exécutée une fois de trop,
– $Q'' = |Q' + \varepsilon\mathcal{M}(\mathcal{B}_1)/m_i + \mathcal{M}(\mathcal{B}_1)|_{\mathcal{M}(\mathcal{B}_2)}$ si la ligne 13 a été exécuté une fois de moins que pour les valeurs non perturbées.

Par conséquent, la valeur $(S)_{\mathcal{B}'}$ a l'entrée du second changement de base devient :

$$S' = |S + \varepsilon p m_i^{-1}|_{\mathcal{M}(\mathcal{B}_2)} \tag{5.8}$$
$$\text{ou } S' = |S + \varepsilon p m_i^{-1} - p|_{\mathcal{M}(\mathcal{B}_2)} \tag{5.9}$$
$$\text{ou } S' = |S + \varepsilon p m_i^{-1} + p|_{\mathcal{M}(\mathcal{B}_2)} \tag{5.10}$$

On considère la détection de faute. Grâce à la précision apportée par le s-Cox, l'équation suivante est vérifiée :

$$\text{Pas de détection de faute} \Rightarrow S' < \mathcal{M}(\mathcal{B}_2)2^{-\sigma} \text{ ou } S' > \mathcal{M}(\mathcal{B}_2)(1-2^{-\sigma-1}). \quad (5.11)$$

Enfin, comme $0 \leq X < 3p < 2^{-\sigma-1}$, l'equation 5.11 devient :

$$\text{Pas de détection de faute} \Rightarrow |X'-X|_{\mathcal{M}(\mathcal{B}_2)} < \mathcal{M}(\mathcal{B}_2)2^{-\sigma} \text{ ou } X' > \mathcal{M}(\mathcal{B}_2)(1-2^{-\sigma}). \quad (5.12)$$

Comme X est entre 0 et $3p$, on a :

$$\forall i \in [0;n[, \forall \epsilon_i \in]-m_i;2^r[, \epsilon_i \notin \{0,m_i\}, \forall \zeta \in \{-1,0,1\}\ 2^{-\sigma} < \frac{|\varepsilon p m_i^{-1} + \zeta p|_{\mathcal{M}(\mathcal{B}_2)}}{\mathcal{M}(\mathcal{B}_2)} < 1-2^{-\sigma}$$
$$\Rightarrow \text{une perturbation est détectée.} \quad (5.13)$$

Je viens de démontrer que toute perturbation sur un canal de $\mathcal{B}$ crée une déviation dépendante de la base $\mathcal{B}$ et de p. Par simple dénombrement, le nombre de perturbations possibles est $6n2^r$. Si on les considère équitablement réparties entre 0 et $\mathcal{M}(\mathcal{B}_1)$, on peut conclure la démonstration. Il faut noter que ce cas est un pire cas : comme p est petit en comparaison de $\mathcal{M}(\mathcal{B}_2)$ les 3 valeurs suivantes $|\epsilon p m_i^{-1} + \zeta p|_{\mathcal{M}(\mathcal{B}_2)}/\mathcal{M}(\mathcal{B}_2)$ pour $\zeta \in \{-1,0,1\}$ sont en réalité proches les une des autres. On peut par ailleurs noter que la vérification de l'inéquation 5.13 peut être exhaustivement testée pour tout i,ϵ_i et ζ.

5.3.6 Perturbation sur le s-Cox ou sur le séquenceur

Je viens ainsi de montrer que si l'on respecte les conditions du théorème 13 (consistant à élever la taille de la base $\mathcal{B}'$, et augmenter la précision du Cox), toute faute unique sur une valeur spécifique manipulée par un Rower sera détectée, tant que la séquence d'opérations n'est pas affectée par la perturbation. En effet, il est nécessaire de réaliser une réduction de $\mathcal{B}'$ vers $\mathcal{B}$. Malheureusement, il n'est pas possible de prouver une telle propriété sur le Cox ou sur le séquenceur. Néanmoins, comme le Cox ne constitue qu'une faible partie du Cox-Rower global, et comme il est nécessaire de protéger le séquenceur contre la faute, il est possible d'utiliser des techniques classiques de protection, comme la redondance ou toute autre technique permettant de détecter les erreurs d'exécution dans cette sous-partie du circuit.

5.3.7 Combinaison des 2 contre-mesures

Naturellement, il est possible d'utiliser sur la même implémentation les 2 contre-mesures. En effet, la nature des conditions sur les bases décrites par les théorèmes 11, 12 et 13 détermine des conditions uniquement sur r, r_ε et n. Par conséquent, il est aisé de choisir une base $\mathcal{B}$ de taille $2n$ telle que pour toute permutation γ, $\mathcal{B}_{1,\gamma}$ et $\mathcal{B}_{2,\gamma}$ respectent les conditions de ces théorèmes. Par ailleurs, je n'ai pas démontré l'absence de faute non détectée sur la base $\mathcal{B}$, mais seulement restreint leur probabilité d'apparition. Néanmoins, on constate en examinant l'équation 5.8, que les possibles perturbations sur la valeur de l'accumulateur du Cox dépendent de p mais aussi $\mathcal{M}(\mathcal{B}_{1,\gamma})$ et $\mathcal{M}(\mathcal{B}_{2,\gamma})$. Ainsi, la randomisation de la base $\mathcal{B}$ permet de compliquer encore la tâche de l'attaquant.

Algorithme 28 : $RSA - CRT$ for RNS (p, q, d_p, d_q, X, m)

Entrée(s): p, q, d_p, d_q une clé, m un message, $\mathcal{B}$ a RNS base,
Entrée(s): `TrMgt`, algorithme 29
Sortie(s): $c = |m^d|_{pq}$
1: `Radix2RNS`$(m, \mathcal{M}(\mathcal{B}))$
2: computes γ a partition of $\mathcal{B}$ in $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$,
3: $M_p =$ `TrMgt`$(m, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
4: $M_q =$ `TrMgt`$(m, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
5: $C_p =$ `expo`$(M_p, d_p, p, , \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
6: $C_q =$ `expo`$(M_q, d_q, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
7: $c_p =$ `RedM`$(C_p * |q^{-1}|_p, p, \mathcal{B}_{2,\gamma}, Bg)$
8: $c_q =$ `RedM`$(C_q * |p^{-1}|_q, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
9: $C = c_p * q + c_q * p$
10: **retourne** `RNS2Radix`(C, M)

5.4 Implémentation d'un RSA-CRT 1024 protégé avec les contre-mesures LRA et RFD

Dans cette section, je propose une implémentation de la primitive de déchiffrement RSA 1024 bits. Cette implémentation utilise un Cox-Rower utilisant les protections LRA et RFD. L'implémentation utilise par ailleurs la technique bien connue du RSA-CRT que j'ai décrite dans la section 1.2.5.

L'algorithme 28 décrit les étapes de calcul adaptées dans un Cox-Rower utilisant les contre-mesures LRA et RFD.

On peut constater que je n'utilise pas la recombinaison classique de Garner [53] (voir algorithme 9), mais la recombinaison proposée utilise l'équation 2.1. En effet, cette version nécessite une multiplication de plus que la recombinaison de Garner, mais elle peut être mixée avec le calcul du représentant de Montgomery inverse. On constate que cette technique ne peut pas être utilisée avec la recombinaison de Garner : $C = c_q + q(|q^{-1}|_p(c_p - c_q))$ qui nécessite une addition supplémentaire.

5.4.1 Dimensionnement du Cox-Rower pour l'exécution du RSA-CRT

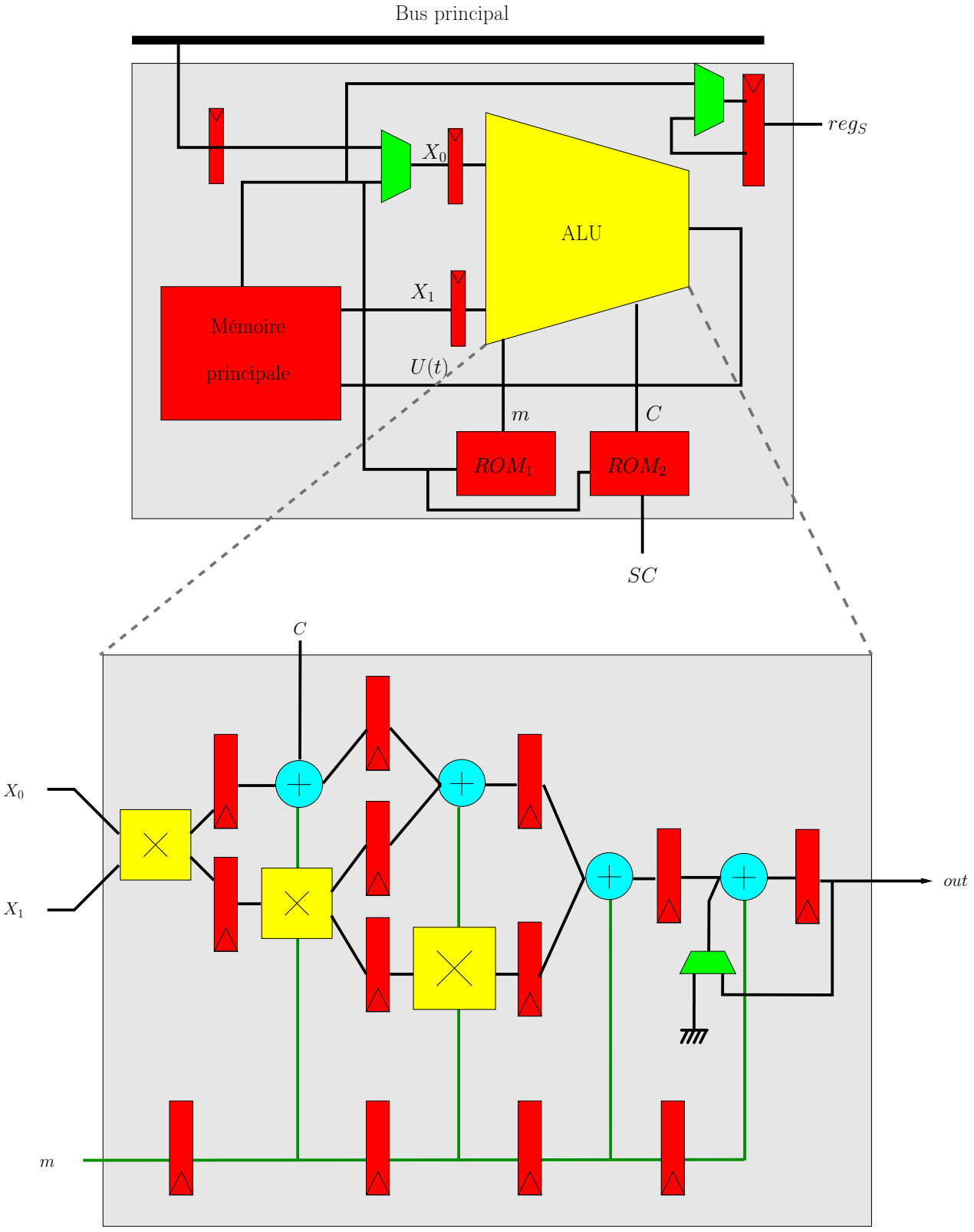
L'utilisation du RSA-CRT ne nécessite aucune réduction par $N = pq$ le module RSA excepté dans l'étape de reconstruction : (comme c_p et c_q peuvent atteindre $3p$, $C < 6N$). Cette étape de réduction ne peut pas être réalisée par un Cox-Rower, mais elle reste relativement simple pour être exécutée à l'extérieur du Cox-Rower par un CPU embarqué de faible puissance. Ainsi, je propose un Cox-Rower avec les paramètres suivants :

n	r	r_ε	ν	s	σ	ϕ
16	36	8	1	2	51	8

Je privilégie un chargement séquentiel des mémoires du Cox-Rower tel que décrit dans la section 5.2. Par conséquent, aucun matériel supplémentaire n'est nécessaire pour le chargement des RAMs. Dans la figure 5.4, j'indique la structure du Rower ainsi que celle du pipeline utilisés. La profondeur du pipeline est $\pi_d = 7$.

On peut remarquer qu'avec ces paramètres, la valeur de σ est restreinte par le niveau du développement limité s et non pas par la taille de $\mathcal{M}(\mathcal{B}_2)$. Mais ces paramètres sont

FIGURE 5.4 – Architecture du Rower et pipeline à 7 étages



suffisants pour s'assurer qu'une perturbation sur un unique Rower sera automatiquement détectée. J'ai synthétisé le Cox-Rower sur le FPGA EP3SL50F484C2, le plus petit FPGA de la série Stratix III (l'outil de synthèse est disponible dans l'édition gratuite du logiciel Quartus II d'Altera). Afin de protéger le Cox et le séquenceur contre les perturbations, j' ai réalisé une simple redondance de ces blocs. Le circuit final est cadencé par une horloge unique qui peut fonctionner jusqu'à 148 MHz à $85°C$, selon le logiciel Altera Quartus II.

Ainsi, le coût total de la contre-mesure LRA et RFD en terme de consommation de ressources est de 1323 ALM sur 8385 (soit 15%), 13 multiplieurs 18×18 sur 148 (9%), et finalement 6 blocs de mémoire M9k sur 34 (18%).

Afin de protéger l'étape d'exponentiation contre les attaques par canaux auxiliaires, j'ai implémenté l'échelle de Montgomery avec la randomisation de l'exposant (d est transformé en $d + a\phi(N)$ où a est un aléa) et la contre-mesure de Giraud [57] (algorithme 10). La contre-mesure de Giraud est ajoutée car elle ne coûte qu'une réduction dans la mesure où l'échelle de Montgomery est utilisée. Grâce à ces contre-mesures, des attaques puissantes telles que l'"address bit DPA" ne peuvent être réalisées.

Ce mode d'exponentiation est nettement moins efficace que la contre-mesure proposée par Rivain [106], mais ne nécessite pas de calcul de la double-chaîne à la volée (ce calcul est nécessairement à la volée du fait de la randomisation de l'exposant). L'adaptation de la contre-mesure proposée par Rivain est néanmoins possible, mais complique le travail de remplissage de pipeline comparé à l'échelle de Montgomery (où pour chacun des 512 bits de d_p et d_q, pas moins de 4 réductions sont nécessaires et parallèlisables). L'utilisation de la contre-mesure LRA permet de contrecarrer les attaques proposées par Fouque [49] et améliorées par Homma et al [68]. Grâce à l'échelle de Montgomery et à la randomisation de l'exposant, je contrecarre les attaques "safe error" (qui nécessitent de pouvoir rejouer le même exposant pour être efficaces).

5.4.2 Calcul du représentant de Montgomery

Considérons l'entrée m de l'algorithme RSA-CRT, (avec $m < pq$). Une fois la transformation RNS $\rightarrow$ binaire réalisée, il est nécessaire de calculer $M(m) = |m\mathcal{M}(\mathcal{B}_{1,\gamma})|_p$. Dans [16], les auteurs proposent l'utilisation de $\texttt{RedM}(m \times |\mathcal{M}(\mathcal{B})|_p, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$ qui donne un résultat correct. Dans la secion 5.2, j'ai démontré que l'utilisation de l'astuce de Bajard était possible avec un Cox-Rower. Néanmoins, le RSA-CRT a une spécificité. En effet, m est 2 fois plus long que p soit 1024 bits dans notre cas. Par conséquent, $m \times |\mathcal{M}(\mathcal{B})|_p$ peut atteindre jusqu'à 1536 bits et il est impossible de représenter un tel nombre dans le Cox-Rower proposé vus les paramétrages de la partie 5.4.1. L'algorithme 29 donne cette adaptation. Elle consiste à réduire immédiatement m avant de réaliser 2 réductions d'affilé. Il faut noter que l'ordre des lignes 1, 3 et 5 n'a pas d'importance.

Algorithme 29 : $\texttt{TrMgt}(m, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$

Entrée(s): $\mathcal{B} = \mathcal{B}_{1,\gamma} \cup \mathcal{B}_{2,\gamma}$ bases RNS, avec p premier, $(X)_{\mathcal{B}}$ un message,
Sortie(s): $(|X\mathcal{B}_{1,\gamma}|_p)_{\mathcal{B}}$

1: $A = \texttt{RedM}(X, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$
2: $B = A \times |\mathcal{M}(\mathcal{B})|_p$
3: $C = \texttt{RedM}(B, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$
4: $D = C \times |\mathcal{M}(\mathcal{B})|_p$
5: $E = \texttt{RedM}(D, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
6: **retourne** E

5.4.3 Reconstruction du CRT et transformation RNS → binaire

Il est aisé de voir qu'avec les contre-mesures LRA et RFD, l'algorithme 28 est protégé contre les attaques par canaux auxiliaires et les fautes jusqu'à la ligne 8. Après, les variables ne sont plus masquées par la contre-mesure LRA. Heureusement, à ma connaissance, il n'existe pas d'attaque par canaux auxiliaires capable d'exploiter cette vulnérabilité (à l'exception éventuelle d'attaques extrêmement performantes comme les attaques template, mais qui nécessitent des conditions très avantageuses d'initialisation et d'exploitation pour l'attaquant). En revanche, l'opération de la ligne 9 est vulnérable aux attaques par perturbation, et il n'existe aucune réduction de Montgomery postérieure à cette opération et permettant une protection par la contre-mesure RFD. Toute attaque postérieure à la ligne 9 est inutile, puisque la valeur manipulée est la sortie de l'exponentiation, et ne trahit donc pas la valeur de la clé privée RSA.

Afin de protéger la ligne 9, je propose l'utilisation de la transformation RNS → binaire proposée dans la section 2.5.2, avec la contre-mesure RFD. En effet, l'idée principale de la reconstruction est de considérer les 2 bases RNS suivantes $\mathcal{B}_1 = \{2^r\}$ et $\mathcal{B}_2 = \mathcal{B} \cup \mathcal{B}' - \{2^r\}$, la base $\mathcal{B}_1$ permettant de calculer le reste de la division de la valeur à transformer X modulo 2^r, et $\mathcal{B}_2$ de retrouver le quotient de la division de X par 2^r.

Les bases $\mathcal{B}_1$ et $\mathcal{B}_2$ ainsi choisies présentent les mêmes propriétés que celles utilisées lors du second changement de base de l'algorithme 11 de réduction de Montgomery. Par conséquent, avec la proposition de paramétrisation du Cox-Rower présentée dans la section 5.4.1, les conditions nécessaires au théorème 12 sont remplies, toute faute sur $\mathcal{B}_2$ sera detectée. Par l'utilisation des formules du théorème 11 et 13, on peut en déduire que $\sigma = 50$. La probabilité qu'il existe une valeur spécifique de X et une perturbation sur $\mathcal{B}_1$ ne conduisant pas à une détection est majorée par 2^{-5}. Par ailleurs, l'utilisation des bases fixes permet de choisir les bases qui permettent de démontrer exhaustivement que l'inéquation 5.13 est toujours correcte quelle que soit la perturbation.

5.4.4 Latence du RSA-CRT

Dans le tableau 5.3 je compare 2 implémentations du RSA-CRT : la première présentée ci-dessus, et la seconde qui est une version utilisant une échelle de Montgomery et une randomisation de l'exposant de 36 bits. La paramétrisation du Cox-Rower est la suivante :

n	r	r_ε	ν	s	σ	ϕ
15	36	8	1	1	25	0

Cette implémentation, dite de référence, n'utilise aucune contre-mesure développée dans ce chapitre. J'utilise cette implémentation pour calculer le coût de nos contre-mesures LRA et RFD. Les résultats obtenus montrent une augmentation générale de 6778 du nombre de cycles nécessaires, soit $7,5\%$ du temps total de calcul, soit 2264 $(2,5\%)$ pour la contre-mesure LRA et 4514 (5%) pour la contre-mesure RFD.

Le temps total de calcul, incluant les 2 contre-mesures est de $0,62$ ms.

5.5 Conclusion

Dans ce chapitre, j'ai proposé une adaptation du Cox-Rower lui apportant une résistance aux attaques par canaux auxiliaires (contre-mesure LRA), et une résistance contre

les attaques par faute (RFD). La première contre-mesure est une adaptation de la contre-mesure de Bajard et al. [16], la deuxième étant nouvelle. Ces deux contre-mesures s'insèrent de manière efficace à la structure initiale du Cox-Rower proposé par [71], en permettant un compromis temps surface très avantageux par rapport à l'implémentation initiale. En outre ces 2 contre-mesures s'adaptent à tout traitement utilisant de l'arithmétique de grands nombres, comme le RSA mais aussi le DSA sur corps premier ou les courbes elliptiques (avec un impact différent dans chaque cas, qui n'a pas été évalué ici). J'ai implémenté un RSA-CRT 1024 bits dans un Cox-Rower utilisant les 2 contre-mesures dans un FPGA, afin de mesurer le coût tant en espace qu'en latence. Les résultats obtenus montrent une augmentation raisonnable (15%) des ressources matérielles supplémentaires, ainsi qu'une augmentation légère de la latence (6%). La suite des travaux doit permettre de mesurer l'efficacité de ces contre-mesures, soit en analysant la qualité du masque apporté par la contre-mesures LRA, soit en analysant des modèles de faute permettant de mettre en défaut la contre-mesure RFD.

5.6 Annexe A : La contre-mesure de Ciet et al.

Dans l'article [37], les auteurs proposent une implémentation FPGA du RSA-CRT. Ils utilisent le Cox-Rower dans une configuration $r = 64$ bits, et réalisent des multiplieurs modulaires séquentiels au lieu des multiplieurs intégrés dans les FPGA, ce qui expliquent entièrement les faibles performances de leur implémentation en comparaison de celle qui est exposée dans la section 5.4.

En revanche, l'intérêt de cet article est que, pour la première fois dans la littérature scientifique le RNS n'est pas simplement vu comme une simple représentation de nombres permettant hypothétiquement d'accélérer le calcul, mais aussi comme une possibilité d'obtenir des contre-mesures efficaces contre les attaques side channel ainsi que les fautes. Notons aussi qu'une idée similaire est proposée dans le brevet [89]. Malheureusement, ces 2 contributions éludent des problématiques et comportent des erreurs que je détaille ici.

La contre-mesure contre les canaux auxiliaires consiste à choisir de manière aléatoire 2 bases de 9 éléments dans un ensemble constitué de 64 valeurs. Ainsi, toute valeur manipulée serait ainsi masquée par la base choisie (voir section 5.2). Néanmoins, ce système de masquage repose sur le choix aléatoire d'une base. Or, comme nous l'avons vu dans le chapitre 2 (tableau 2.1), de nombreux précalculs sont nécessaires à l'exécution dans une base RNS d'un traitement algorithmique quel qu'il soit. Ce nombre de précalculs est essentiellement dépendant du choix de l'algorithme de changement de base. Or, l'algorithme limitant au maximum les précalculs nécessaires au changement de base est l'algorithme 12 qui utilise la représentation MRS. Les précalculs nécessaires sont alors les seuls $\mu_{i,j} = |{m_j}^{-1}|_{m_i}$. Il faut ainsi noter que cela représente pour la contre-mesure de Ciet et al. [37] pas moins de 4032 précalculs, soit $31,5$ ko de précalculs à stocker. Par ailleurs, il est impossible d'utiliser dans ce cas le Cox-Rower, qui ne permet pas d'implémenter efficacement cet algorithme, à cause d'un parallélisme limité. Par ailleurs, le problème du calcul de $|\mathcal{M}(\mathcal{B})^2|_p$ pour la transformation en représentant de Montgomery n'est pas traité, et dans la mesure où cette valeur dépend du choix de la base, un stockage naïf nécessite ainsi environ 2^{59} valeurs de 64 bits, soit 4 millions de Teraoctets. Par conséquent, il est nécessaire de trouver un moyen de calculer cette valeur de manière efficace en limitant les précalculs.

Dans le même article, les auteurs proposent par ailleurs une contre-mesure contre les fautes. L'idée simple de cette contre-mesure est l'utilisation d'un canal supplémentaire dans la base RNS servant à l'exponentiation. Pour détecter une erreur, les auteurs proposent que la valeur m_r du canal soit supérieure à l'ensemble des valeurs des autres canaux. Ainsi, en réalisant un changement de base entre $(m_1; \cdots; m_n)$ et (m_r), une valeur identique doit être retrouvée. Une perturbation sur un autre canal de la base sera automatiquement détectée. Cet algorithme nécessite donc un changement de base, soit un traitement correspondant à $2n$ multiplications modulaires, et n additions, qui doivent être réalisées séquentiellement. Les auteurs de [37] n'indiquent pas quand est réalisé ce traitement. Or, cette indication est essentielle puisqu'elle influe et sur la vitesse de traitement, et sur l'efficacité de la contre-mesure. En effet, il est assez facile de mettre en évidence que si $\mathcal{M}(\mathcal{B})$ est largement supérieure à p (par exemple avec l'utilisation d'un canal supplémentaire), et que si $\mathcal{B}'$ est d'une taille équivalente à $\mathcal{B}$, alors pour n'importe quelle valeur aléatoirement prise entre 0 et $\mathcal{M}(B)\mathcal{M}(\mathcal{B}')$, plusieurs tours de mise au carré de cette valeur ,en utilisant l'algorithme de réduction de Montgomery, réduit progressivement sa valeur entre 0 et $3p$. La vitesse de dépréciation de la suite dépend alors du rapport entre p et $\mathcal{M}(\mathcal{B})$. Une étude approfondie est nécessaire pour déterminer les conditions pour lesquelles une contre-mesure se reposant uniquement sur la comparaison de la valeur finale avec un résultat est une contre-mesure apportant une garantie.

TABLE 5.2 – Consommation des ressources du FPGA

Bloc	nombre	logique	18×18 DSP	Mémoire
Rower	16	467 ALM	9	2 M9k
Cox	2	220 ALM	4	0
Séquenceur	2	416 ALM	0	2 M9k
Cox-Rower	1	8385 ALM	148	34 M9k

TABLE 5.3 – Temps de calcul pour un RSA-CRT avec et sans les contre-mesures

.

Sous fonctions	RSA-CRT sans contre-mesures	RSA-CRT avec contre-mesures	coût supplé
Chargement des RAM	0	1806	1806
Précalcul des bases	0	242	242
`Radix2RNS`	254	254	0
`TrMgt`	80	296	216
`expo`	82771	87155	4384
Reconstruction CRT	93	97	4
`RNS2Radix`	1073	1209	136
total	84271	91059	6778

Conclusion

La conclusion principale de cette thèse est que pour toute implémentation matérielle de primitives sur courbes elliptiques de grande caractéristique recherchant à la fois flexibilité, vélocité et sécurité , tout en restant dans un contexte de consommation de ressources raisonnable, le Residue Number System est compétitif. Le compromis entre flexibilité, performance et sécurité est interessant, le compromis entre ces deux derniers objectifs étant moins pénalisant que d'autres approches. De plus les spécificités du RNS et de son mode de représentation peuvent rester contenus dans l'architecture Cox-Rower pour l'ensemble des primitives étudiés dans cette thèse (RSA, ECC ou couplages), ce qui permet une intégration harmonieuse dans un système plus complet comme un System On Chip.

Le chapitre 3 a démontré pour les courbes elliptiques génériques que le RNS permettait sur de nombreuses technologies d'obtenir une latence raisonnable. De nombreuses pistes d'amélioration restent cependant à étudier, notamment l'étude de courbes spécifiques (Hessian, Edwards, Montgomery, genres supérieurs, etc.) pour trouver le meilleurs compromis temps/sécurité à chaque niveau de sécurité, dans un cadre expérimental (en prenant en compte la dépendance de données, l'utilisation de la réduction fainéante, etc.). Le développement sur un composant spécifique de dernière génération devrait par ailleurs permettre d'obtenir une augmentation de la fréquence, et l'utilisation de bases spécifiques comme celle proposée par [121, 17] pourrait apporter un gain en nombres de cycles. Enfin, dans la contribution [52], les auteurs proposent une adaptation de la représentation de Montgomery permettant de limiter la complexité du RNS de $2n^2 + 3n$ à $2n^2 + 2n$, mais qui n'est proposée que pour une exponentiation dans $\mathbb{F}_p$. Une étude de son portage dans le cadre des courbes elliptiques est à réaliser.

Dans le chapitre 4, j'ai démontré que le RNS associé aux courbes de Barreto-Naehrig est aujourd'hui une solution matérielle très efficace pour le couplage au niveau de sécurité de 128 bits. Elle surpasse les implémentations FPGA en représentation binaire mais est compétitive avec les implémentations en petite caractéristique qui sont handicapées par leur faible degré de plongement, et qui nécessitent du coup beaucoup de ressources matérielles pour atteindre des résultats équivalents. Une limite semble ainsi se dessiner pour le choix des courbes pour le couplage : à moins de 128 bits, les courbes supersingulières en caractéristique 2 et 3 surpassent la grande caractéristique,et au-delà la grande caractéristique est meilleure. Par ailleurs, le fossé entre les implémentations logicielles et matérielles s'est considérablement amoindri grâce aux travaux de cette thèse et à la contribution [35]. La capacité de parallélisme des FPGA et des implémentations matérielles étant très loin d'être complètement exploitée, il est nécessaire d'approfondir le parallélisme de l'algorithme de couplage, comme [24] l'a fait pour la petite caractéristique. Enfin, une étude comparative des différentes familles de courbes "pairing friendly" potentielles pour le calcul de couplage à haut niveau de sécurité (au delà de 192 bits), tant au niveau algorithmique (recherche de spécificité des calculs, choix de couplage, recherche de courbes avec une représentation efficace du corps sous-jacent) qu'au niveau implémentation. Par exemple, les courbes KSS [79],

qui proposent un degré de plongement supérieur ($k = 18$) ont des atouts indéniables à ces niveaux. En revanche, elles sont handicapées par $\rho = 1.33$ (contre 1 pour les courbes BN). L'implémentation de la courbe BN_{192} proposée est ainsi un premier point de comparaison dans ce domaine.

Enfin, le chapitre 5 a proposé une implémentation de la contre-mesure proposée par [16] ainsi qu'une nouvelle contre-mesure permettant de détecter les pannes, erreurs d'exécutions, et bien sûr les perturbations volontaires d'un attaquant. Par cela, j'ai démontré que le RNS permettait une couverture contre les attaques par canaux auxiliaire et par perturbation à un prix extrêmement raisonnable, et ce indépendamment de la primitive cryptographique employée, pourvu qu'elle utilise l'arithmétique de grands nombres. Par exemple, dans le cas du RSA-CRT 1024 bits, ce surcoût est respectivement de $7,5\%$ en temps et 15% en matériel. Un problème ouvert est l'étude de la sécurité intrinsèque de ces contre-mesures. Dans quelle conditions le masquage et la contre-mesure contre les perturbations proposée sont-ils réellement efficaces ? L'efficacité de ces contre-mesure peut-elle être améliorée par un choix de bases approprié ?

Bibliographie

[1] Altera web site. http://www.altera.com.

[2] Cadence design systems web site. http://www.cadence.com.

[3] GMP : the GNU Multiple Precision Library. http://gmplib.org.

[4] IEEE 1164 standard multivalue logic system for VHDL model interoperability. http://en.wikipedia.org/wiki/IEEE_1164.

[5] The IEEE Verilog 1364-2001 standard. http://www.verilog.com/IEEEVerilog.html.

[6] The IEEE VHDL 1076 standard. http://en.wikipedia.org/wiki/IEEE_1076.

[7] *The Stratix III handbook*. http://www.altera.com/literature/lit-stx3.jsp.

[8] Xilinx web site. http://www.xilinx.com.

[9] Jithra Adikari, Vassil S. Dimitrov, and Laurent Imbert. Hybrid binary-ternary number system for elliptic curve cryptosystems. *IEEE Transactions on Computers*, 60(2) :254–265, 2011.

[10] D. Aranha, J.-L. Beuchat, J. Detrey, and N. Estibals. Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves. Cryptology ePrint Archive, Report 2010/559, 2010. http://eprint.iacr.org/.

[11] D. Aranha, J. López, and D. Hankerson. High-speed parallel software implementation of the η_T pairing. In *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *LNCS*, pages 89–105. Springer, Heidelberg, 2010.

[12] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In *EUROCRYPT*, pages 48–68, 2011.

[13] J.C. Bajard, S. Duquesne, and M. Ercegovac. Combining leak–resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$ and RNS representation. Cryptology ePrint Archive, Report 2010/311, 2010. http://eprint.iacr.org/.

[14] Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. An RNS Montgomery modular multiplication algorithm. *IEEE Trans. Computers*, 47(7) :766–776, 1998.

[15] Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. Modular multiplication and base extensions in residue number systems. In *IEEE Symposium on Computer Arithmetic*, pages 59–65, 2001.

[16] Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak resistant arithmetic. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 116–145. Springer Berlin / Heidelberg, 2004.

[17] Jean-Claude Bajard, Marcelo E. Kaihara, and Thomas Plantard. Selected RNS bases for modular multiplication. In *IEEE Symposium on Computer Arithmetic*, pages 25–32, 2009.

[18] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, pages 354–368, London, UK, 2002. Springer-Verlag.

[19] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of SAC 2005, volume 3897 of LNCS*, pages 319–331. Springer-Verlag, 2005.

[20] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 311–323, London, UK, 1987. Springer-Verlag.

[21] Daniel J. Bernstein. Cache-timing attacks on AES. Technical report, 2005.

[22] Alexandre Berzati, Cécile Canovas, and Louis Goubin. Perturbating RSA public keys : An improved attack. In *CHES*, pages 380–395, 2008.

[23] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core implementation of the Tate pairing over supersingular elliptic curves. In *Cryptology and Network Security*, volume 5888 of *LNCS*, pages 413–432. Springer, Heidelberg, 2009.

[24] Jean-Luc Beuchat, Jérémie Detrey, Nicolas Estibals, Eiji Okamoto, and Francisco Rodríguez-Henríquez. Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves. *IEEE Trans. Computers*, 60(2) :266–281, 2011.

[25] J.L. Beuchat, J.E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In *Pairing*, volume 6487 of *LNCS*, pages 21–39, 2010.

[26] Johannes Blomer, Martin Otto, and Jean pierre Seifert. A new CRT-RSA algorithm secure against Bellcore attacks. In *CCS 2003, ACM SIGSAC, ACM Press*, pages 311–320. ACM Press, 2003.

[27] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.

[28] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.

[29] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. pages 514–532. Springer-Verlag, 2001.

[30] Arnaud Boscher, Robert Naciri, and Emmanuel Prouff. CRT RSA algorithm protected against fault attacks. In *Proceedings of the 1st IFIP TC6 /WG8.8 /WG11.2 international conference on Information security theory and practices : smart cards, mobile and ubiquitous computing systems*, WISTP'07, pages 229–243, Berlin, Heidelberg, 2007. Springer-Verlag.

[31] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg.

[32] Eric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography*, pages 335–345, 2002.

[33] Çetin Kaya Koç and Colin D. Walter. Montgomery arithmetic. In *Encyclopedia of Cryptography and Security*. 2005.

[34] Suresh Chari, Josyula Rao, and Pankaj Rohatgi. Template attacks. In Burton Kaliski, Çetin Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg.

[35] Ray Cheung, Sylvain Duquesne, Junfeng Fan, Nicolas Guillermin, Ingrid Verbauwhede, and Yao Gavin. FPGA implementations of pairing using residue number system and lazy reduction. In *CHES*, 2011.

[36] Mathieu Ciet and Marc Joye. Practical fault countermeasures for chinese remaindering based RSA (extended abstract). In *IN PROC. FDTC'05*, pages 124–131.

[37] Mathieu Ciet, Michael Neve, Eric Peeters, and Jean-Jacques Quisquater. Parallel FPGA implementation of RSA with residue number systems – can side-channel threats be avoided. In 46^{th} *International Midwest Symposium on Circuits and Systems : MWSCAS '03*, 2003.

[38] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES*, pages 292–302, 1999.

[39] Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault attacks and countermeasures on vigilant's RSA-CRT algorithm. In *FDTC*, pages 89–96, 2010.

[40] Craig Costello, Tanja Lange, and Michael Naehrig. Faster pairing computations on curves with high-degree twists. In *Public Key Cryptography*, pages 224–242, 2010.

[41] Guerric Meurice de Dormale and Jean-Jacques Quisquater. High-speed hardware implementations of elliptic curve cryptography : A survey. *Journal of Systems Architecture*, 53(2-3) :72–84, 2007.

[42] Xinjun Du, Ying Wang, Jianhua Ge, and Yumin Wang. An ID-based broadcast encryption scheme for key distribution. *IEEE Transactions on Broadcasting*, 51(2) :264–266, 2005.

[43] S. Duquesne. RNS arithmetic in F_{p^k} and application to fast pairing computation. Cryptology ePrint Archive, Report 2010/555, 2010. `http://eprint.iacr.org/`.

[44] Sylvain Duquesne and Nicolas Guillermin. A FPGA pairing implementation using the residue number system. Cryptology ePrint Archive, Report 2011/176, 2011. `http://eprint.iacr.org/`.

[45] Nicolas Estibals. Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves. In *Pairing*, pages 397–416, 2010.

[46] J. Fan, F. Vercauteren, and I. Verbauwhede. Efficient hardware implementation of $\mathbb{F}_p$-arithmetic for pairing-friendly curves. *IEEE Transactions on Computers*, PP(99) :1, 2011.

[47] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems–CHES 2009*, pages 240–253, 2009.

[48] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. Fault Attack on elliptic curve with Montgomery ladder implementation. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography FDTC*, pages 92–98, Washington DC, États-Unis. IEEE Computer Society Press.

[49] Pierre-Alain Fouque and Frédéric Valette. The doubling attack - *why upwards is better than downwards*. In *CHES*, pages 269–280, 2003.

[50] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *J. Cryptology*, 23(2) :224–280, 2010.

[51] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput.*, 62 :865–874, April 1994.

[52] Filippo Gandino, Fabrizio Lamberti, Paolo Montuschi, and Jean-Claude Bajard. A general approach for improving RNS Montgomery exponentiation using preprocessing. In *IEEE Symposium on Computer Arithmetic*, pages 195–204, 2011.

[53] Harvey L. Garner. The residue number system. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, IRE-AIEE-ACM '59 (Western), pages 146–153, New York, NY, USA, 1959. ACM.

[54] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury. High speed flexible pairing cryptoprocessor on FPGA platform. In *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *LNCS*, pages 450–466. Springer, Heidelberg, 2010.

[55] Santosh Ghosh, Dipanwita Roy Chowdhury, and Abhijit Das. High speed cryptoprocessor for η_T pairing on 128-bit secure supersingular elliptic curves over characteristic two fields. In *CHES*, pages 442–458, 2011.

[56] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *CHES*, pages 426–442, 2008.

[57] Christophe Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. Computers*, 55(9) :1116–1120, 2006.

[58] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In Yvo Desmedt, editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 199–211. Springer Berlin / Heidelberg, 2002.

[59] P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 35–50. Springer, Heidelberg, 2009.

[60] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In Phong Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer Berlin / Heidelberg.

[61] Nicolas Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. In *CHES*, pages 48–64, 2010.

[62] Nicolas Guillermin. A coprocessor for secure and high speed modular arithmetic. Cryptology ePrint Archive, Report 2011/354, 2011. `http://eprint.iacr.org/`.

[63] Tim Güneysu and Christof Paar. Ultra high performance ecc over nist primes on commercial FPGAs. In *CHES*, pages 62–78, 2008.

[64] D.R. Hankerson, S.A. Vanstone, and A.J. Menezes. *Guide to elliptic curve cryptography*. Springer professional computing. Springer, 2004.

[65] William Hasenplaugh, Gunnar Gaubatz, and Vinodh Gopal. Fast modular reduction. In *IEEE Symposium on Computer Arithmetic*, pages 225–229, 2007.

[66] Florian Hess. Efficient identity based signature schemes based on pairings. In *Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, SAC '02, pages 310–324, London, UK, UK, 2003. Springer-Verlag.

[67] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10) :4595–4602, 2006.

[68] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In *CHES*, pages 15–29, 2008.

[69] Charles Hymans. Checking safety properties of behavioral VHDL descriptions by abstract interpretation. In *SAS*, pages 444–460, 2002.

[70] Charles Hymans. Design and implementation of an abstract interpreter for VHDL. In *CHARME*, pages 263–269, 2003.

[71] Shin ichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-Rower architecture for fast parallel Montgomery multiplication. In *EUROCRYPT*, pages 523–538, 2000.

[72] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In Burton Kaliski, Çetin Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 399–412. Springer Berlin / Heidelberg, 2003.

[73] Tetsuya Izu and Tsuyoshi Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In *Public Key Cryptography*, pages 280–296, 2002.

[74] Ludovic Jacomme, Frédéric Pétrot, and Rajesh K. Bawa. Formal extraction of memorizing elements for sequential VHDL synthesis. In *EUROMICRO*, pages 10317–10620, 1998.

[75] Kimmo U. Järvinen and Jorma O. Skyttä. High-speed elliptic curve cryptography accelerator for koblitz curves. In *FCCM*, pages 109–118, 2008.

[76] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS*, pages 385–394, 2000.

[77] M. Joye and G. Neven. *Identity-based cryptography*. Cryptology and information security series. IOS Press, 2009.

[78] Marc Joye and Sung-Ming Yen. The Montgomery powering ladder. In *CHES*, pages 291–302, 2002.

[79] Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott. Constructing Brezing-Weng pairing friendly elliptic curves using elements in the cyclotomic field. *IACR Cryptology ePrint Archive*, 2007 :452, 2007.

[80] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar. Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, volume 5747 of *LNCS*, pages 254–271. Springer, Heidelberg, 2009.

[81] Koray Karabina. Squaring in cyclotomic subgroups. Cryptology ePrint Archive, Report 2010/542, 2010. `http://eprint.iacr.org/`.

[82] Chris Karlof, David Wagner, Chris Karlof, and David Wagner. Hidden Markov model cryptanalysis. In *In Cryptographic Hardware and Embedded Systems – CHES '03*, pages 17–30. Springer-Verlag, 2003.

[83] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In *CRYPTO*, pages 333–350, 2010.

[84] Donald E. Knuth. *The Art of Computer Programming, Volume I : Fundamental Algorithms*. Addison-Wesley, 1968.

[85] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. In *Proceedings of Cryptography and Coding 2005, volume 3796 of LNCS*, pages 13–36. Springer-Verlag, 2005.

[86] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[87] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[88] J.L. Lagrange and L. Poinsot. *Traité de la résolution des équations numériques de tous les degrés : avec des notes sur plusieurs points de la théorie des équations algébriques*. Bachelier, 1826.

[89] Pierre-Yvan Liardet. Brevet fr20020011671 20020920 : Masquage de donnes décomposées dans un système de résidus. Technical report.

[90] Patrick Longa and Catherine H. Gebotys. Efficient techniques for high-speed elliptic curve cryptography. In *CHES*, pages 80–94, 2010.

[91] David Lubicz and Thomas Sirvent. Attribute-based broadcast encryption scheme made efficient. In *AFRICACRYPT*, pages 325–342, 2008.

[92] George H. Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5) :1045–1079, 1955.

[93] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 80–89, New York, NY, USA, 1991. ACM.

[94] Nele Mentens. *Secure and Efficient Coprocessor Design for Cryptographic Applications on FPGAs*. PhD thesis, KU Leuven, 2007.

[95] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *CHES*, pages 144–157, 1999.

[96] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170) :519–521, 1985.

[97] Edward F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.

[98] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *LATINCRYPT*, pages 109–123, 2010.

[99] Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shin ichi Kawamura. Implementation of RSA algorithm based on RNS Montgomery multiplication. In *CHES*, pages 364–376, 2001.

[100] National Institute of Science and Technology. The digital signature standard. Technical report, http ://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[101] National Institute of Standard and technology. Key management, 2007. `http://csrc.nist.gov/groups/ST/toolkit/key\_management.html`.

[102] White Paper. Stratix vs. Virtex-II pro FPGA performance analysis. Technical report, www.altera.com/literature/wp/wpstxvrtxII.pdf.

[103] Geovandro C. C. F. Pereira, Marcos A. Simplício, Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly bn elliptic curves. *J. Syst. Softw.*, 84 :1319–1326, August 2011.

[104] Karl C. Posch and Reinhard Posch. Modulo reduction in residue number systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(5) :449–454, 1995.

[105] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA) : Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards : Smart Card Programming and Security*, E-SMART '01, pages 200–210, London, UK, UK, 2001. Springer-Verlag.

[106] Matthieu Rivain. Securing RSA against fault analysis by double addition chain exponentiation. In *CT-RSA*, pages 459–480, 2009.

[107] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21 :120–126, February 1978.

[108] Kazuo Sakiyama, Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems. In *ARC*, pages 347–357, 2006.

[109] Akashi Satoh and Kohji Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52 :449–460, 2003.

[110] Dimitrios M. Schinianakis, Apostolos P. Fournaris, Athanasios Kakarountas, and Thanos Stouraitis. An RNS architecture of an F_p elliptic curve point multiplier. In *ISCAS*, 2006.

[111] Dimitrios M. Schinianakis and Thanos Stouraitis. A RNS Montgomery multiplication architecture. In *ISCAS*, pages 1167–1170, 2011.

[112] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography*, Pairing '09, pages 78–88, Berlin, Heidelberg, 2009. Springer-Verlag.

[113] Adi Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks, 1998.

[114] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Trans. Computers*, 38(2) :292–297, 1989.

[115] J.H. Silverman. *The arithmetic of elliptic curves*. Graduate texts in mathematics. Springer-Verlag, 1986.

[116] Nicholas S. Szabo and Richard I. Tanaka. Residue arithmetic and its application to computer technology. 11(1) :103–104, 1969.

[117] Robert Szerwinski and Tim Güneysu. Exploiting the power of GPUs for asymmetric cryptography. In *CHES*, pages 79–99, 2008.

[118] Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1) :455–461, 2010.

[119] David Vigilant. RSA with CRT : A new cost-effective solution to thwart fault attacks. In *CHES*, pages 130–145, 2008.

[120] David Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *ACM Conference on Computer and Communications Security*, pages 92–97, 2004.

[121] Gavin Xiaoxu Yao, Junfeng Fan, Ray C.C. Cheung, and Ingrid Verbauwhede. A high speed pairing coprocessor using RNS and lazy reduction. Cryptology ePrint Archive, Report 2011/258, 2011. `http://eprint.iacr.org/`.

[122] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49 :967–970, 2000.

[123] Fangguo Zhang and Kwangjo Kim. Id-based blind signature and ring signature from pairings. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security : Advances in Cryptology*, ASIACRYPT '02, pages 533–547, London, UK, UK, 2002. Springer-Verlag.